

AFRL-RI-RS-TR-2008-92
In House Final Technical Report
March 2008



100X JOINT BATTLESPACE INFOSPHERE (JBI)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2008-92 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

/s/

GEORGE O. RAMSEYER
Work Unit Manager

JAMES A. COLLINS, Deputy Chief
Advanced Computing Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.</small>					
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) MAR 2008		2. REPORT TYPE Final		3. DATES COVERED (From - To) Nov 03 – Sep 07	
4. TITLE AND SUBTITLE 100X JOINT BATTLESPACE INFOSPHERE (JBI)				5a. CONTRACT NUMBER In House	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER 62702F	
6. AUTHOR(S) George O. Ramseyer, Lok Kwong-Yan, Richard W. Linderman				5d. PROJECT NUMBER 459T	
				5e. TASK NUMBER XJ	
				5f. WORK UNIT NUMBER BI	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFRL/RITB 525 Brooks Rd Rome NY 13441-4505				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RITB 525 Brooks Rd Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2008-92	
12. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 08-0964					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT A high-performance information management architecture, based upon the JBI reference implementation, was developed and tested. The JBI reference implementation, developed at AFRL, specifies core services and a Common API (Application Program Interface) (CAPI) for a network-centric platform to support Command and Control communications. The 100X JBI meets the conceptual goals of the JBI by implementing the JBI CAPI and adhering to the standards established for the JBI core services. The 100X JBI was used in several experiments to evaluate its performance and test the CAPI and core services implementation. This report summarizes the experience in developing the 100X JBI reference implementation and the results of performance experiments.					
15. SUBJECT TERMS Information management, JBI, Joint Battlespace Infosphere, high performance computing, field programmable gate arrays, Command and Control					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 74	19a. NAME OF RESPONSIBLE PERSON George O. Ramseyer
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

TABLE OF CONTENTS

	Page
LIST OF FIGURES.....	iv
LIST OF TABLES.....	v
ABSTRACT.....	vi
1. EXECUTIVE SUMMARY.....	1
2. JBI OVERVIEW	3
2.1 Introduction.....	3
2.1.1 Information Objects.....	3
2.1.2 Interactions of Information Objects with JBI Servers.....	5
2.1.3 Malicious Code Embedded in Information Objects.....	6
2.2 JBI Services.....	7
2.2.1 Web Administration Services.....	7
2.2.2 Authentication Service.....	9
2.2.3 Metadata Repository Service.....	10
2.2.4 Publish and Subscribe Services.....	10
2.2.5 Query Services.....	10
3. 100X JBI SECURITY.....	11
3.1 Secure Shell Tunneling.....	11
3.2 Information Objects.....	12
3.3 Authentication.....	13
3.3.1 Web Administration.....	14
3.3.2 Users.....	14
3.3.3 The Publish, Subscribe, Query and MDR.....	14
3.3.4 Kerberos.....	14
4. 100X JBI ARCHITECTURE.....	16
4.1 Communications.....	17
4.2 Publishing.....	18

4.3	Subscribing.....	18
4.4	Brokering/Disseminating.....	18
4.5	JB1 Connectors.....	19
4.6	Query and Archive.....	19
4.7	Architecture.....	19
5	100X JB1 SYSTEM SEQUENCE DIAGRAMS.....	21
5.1	Publication.....	21
5.2	Subscribe.....	23
5.3	Query.....	25
5.4	Meta Data Repository.....	27
6	DISTRIBUTED INTERACTIVE HPC SYSTEM.....	30
6.1	Overview.....	30
6.2	Linux Clusters.....	30
6.2.1	Mach 2.....	32
6.2.2	Powell.....	32
6.2.3	Koa.....	33
6.2.4	Coyote.....	33
6.2.5	Heterogeneous HPC.....	33
6.2.6	Seafarer.....	33
6.3	Testbed Applications.....	33
7	CORE SERVICE EXPERIMENTS.....	35
7.1	Publisher Catcher Capacity.....	35
7.1.1	Experiment.....	36
7.1.2	Results.....	38
7.2	End to End Single Clause Latency.....	39
7.2.1	Experiment.....	39
7.2.2	Results.....	40
7.3	Predicate Complexity.....	42
7.3.1	Experiment.....	42

	7.3.2	Results.....	44
7.4		100X JBI Throughput.....	45
	7.4.1	Experiment.....	45
	7.4.2	Results.....	48
8		ALPHA TESTING OF THE 100X JBI.....	51
	8.1	Overview.....	51
	8.2	Testing Parameters.....	51
		8.2.1 Software Configuration.....	52
		8.2.2 Hardware Configuration.....	52
		8.2.3 Software Testing Tools.....	52
	8.3	Summary of the Testing Results.....	55
		8.3.1 100X JBI Software Installation.....	55
		8.3.2 Static Code Analysis and Memory Profiling.....	56
		8.3.3 Stress Testing.....	57
		8.3.4 Direct Attacks.....	57
	8.4	Towards Beta Testing.....	58
9		DEMONSTRATIONS.....	59
	9.1	Swatchbuckler.....	59
	9.2	Angel Fire.....	59
	9.3	Paradigms for Parallel Computing.....	60
10		YFILTER BROKERING.....	61
11		SUMMARY.....	62
12		REFERENCES.....	63
		APPENDIX Symbols, Abbreviations and Acronyms.....	64

LIST OF FIGURES

	Page
Figure 1. Metadata Only Information Object.....	4
Figure 2. Metadata and Payload Information Object.....	4
Figure 3. SSH Tunneling.....	12
Figure 4. 100X JBI Data Flow Diagram.....	13
Figure 5. Overview of the 100X JBI architecture.....	16
Figure 6. Detailed Overview of the 100X JBI architecture.....	20
Figure 7. Publish System Sequence Diagram.....	22
Figure 8. Subscribe System Sequence Diagram.....	24
Figure 9. Query System Sequence Diagram.....	26
Figure 10. Meta Data Repository System Sequence Diagram.....	29
Figure 11. Distributed Interactive HPC Testbed.....	31
Figure 12. 100X JBI Architecture Publisher Catcher Capacity.....	35
Figure 13. Information Objects Processed v. Number of Brokers.....	38
Figure 14. 100X JBI Architecture for end to end latency per predicate with a single processor.....	40
Figure 15. End to end latency v. the number of subscribers (predicate clauses).....	41
Figure 16. Architecture for end to end latency per clause with a single processor.....	43
Figure 17. Predicate Complexity vs. End to End Latency.....	44
Figure 18. Architecture for Throughput of 100X JBI.....	47
Figure 19. 100X JBI Throughput Results.....	48
Figure 20. 100X JBI scalability.....	49
Figure 21. Speedup of the 100X JBI verses the JBI 1.2 Reference Implementation.....	49

LIST OF TABLES

	Page
Table 1. JBI Web Administration Service Commands.....	8
Table 2. JBI Authentication Service Commands.....	9
Table 3. DIHT Linux Clusters.....	31
Table 4. Publisher Catcher Parameters	36
Table 5. Publisher Catcher Experimental Conditions.....	37
Table 6. End to End Latency Parameters.....	39
Table 7. Broker Latency Complexity Parameters.....	42
Table 8. Throughput Parameters.....	46
Table 9. Throughput Experimental Conditions.....	46

ABSTRACT

The goal of the 100X JBI project is to develop a high-performance information management architecture that is based upon the JBI reference implementation. The JBI reference implementation, developed at AFRL, specifies core services and a Common API (Application Program Interface) (CAPI) for a network-centric platform to support Command and Control communications. The 100X JBI meets the conceptual goals of the JBI by implementing the JBI CAPI and adhering to the standards established for the JBI core services. The 100X JBI was used in several experiments to evaluate its performance and test the CAPI and core services implementation. This report summarizes the experience in developing the 100X JBI reference implementation and the results of performance experiments.

1. EXECUTIVE SUMMARY

This research addressed numerous technological challenges in developing the necessary methodologies and tools for a speedup of the core services of the Reference Implementation of the Joint Battlespace Infosphere (JBI). In the current world environment, the amount of available information has dramatically increased, compounding the problem of providing the right information to the right people in a timely and accessible fashion. Potential adversaries have access to much of the same information that we do. This dynamic world environment has established the necessity that we surpass our adversaries in the timeliness that information resources can be leveraged. This research investigated techniques that accelerated and scaled the baseline JBI Reference Implementation by 100 times through the use of high performance computers, parallel programming techniques and optimizations.

It was the goal that the 100 X JBI optimized the performance of publish and subscribe services that are required for, as an example, an Air Operations Center (AOC). Such a system will allow either a 100 fold increase in the amount of information to be processed by the JBI core services in the same amount of time, a 100 fold decrease in the amount of time necessary to process the same amount of core information, or some weighted combination of both. The purpose of this system was to accelerate the core services of the Joint Battlespace Infosphere by two orders of magnitude so that the JBI is scaled for use in a warfighting environment. The JBI is an information management system which can be tailored to provide the right information to the warfighter at the right time, without at the same time overwhelming the warfighter with extraneous information.

The system was tested utilizing the Defense Research and Engineering Network (DREN), over which remote high performance computers were accessed in an interactive mode. Two local high performance computers, Coyote and the Heterogeneous High Performance Computer (HHPC), were accessed, as well as Seahawk at the Space and Naval Warfare Center, San Diego (SSCSD), Powell at the Army Research Laboratory (ARL), Mach 2 at ASC WPAFB, and the Koa Cluster at the Maui High Performance

Computing Center (MHPCC). The accelerated core services of the JBI were demonstrated within this distributed environment.

The system was alpha tested, and a summary of the results are presented here. The purpose of testing is to find faults, and revisions to the software were made during testing to correct some of the faults. Other changes to the system will be made before beta testing, while some of the faults are beyond the scope of this effort, and would need to be incorporated at the reference implementation level.

YFiltering has been explored as a way to rapidly broker information objects. A brief summary of that work is presented here, and a come complete description will be published in an additional government document. YFiltering is being integrated with the 100X JBI system, and additionally with the 100K Infosphere effort, an on-going 6.2 effort here at the Information Directorate. The 100K Infosphere effort is also integrating the initial field programmable gate array work presented here into that information management system.

2. JBI OVERVIEW

An overview of the Joint Battlespace Infosphere (JBI) is presented in this section.

2.1 Introduction

The JBI is an information management system based on the publish/subscribe distributed systems paradigm. The JBI provides a means for a Community of Interest (COI) to share information in a centralized and organized fashion. JBI clients are separated into two main groups, the publishers (producers of information) and the subscribers (consumers of information). The users share information using data primitives called information objects. Additional information on the JBI can be found in the Reference Implementation Quick Start Guide (1).

2.1.1 Information Objects

An information object consists of two components, the metadata and the payload. Information objects are analogous to well-defined emails. Emails can also be separated into two main parts, a message text body and data attachments. The main difference between information objects and emails is that information objects have an exact format that must be followed.

Metadata is written as an Extensible Markup Language (XML) document that describes the content, quality, condition, and other characteristics of data, which is in the payload, represented in an information object, and are often described as “data about data.” Metadata may or may not have a payload associated with it. In Figure 1 is presented a metadata weather related information object.

```

<metadata>
  <weather>
    <temperature>
      <value>50</value>
      <scale>F</scale>
    </temperature>
    <location>
      <city>Kansas City</city>
      <state>Kansas</state>
    </location>
  </weather>
</metadata>

```

Figure 1. Metadata Only Information Object

When the information object has a payload associated with it, then the data in the payload must have an exactly defined format. This metadata then describes the payload (i.e. file type and encoding) and any other related information. Typical formats include the exact type (e.g. image, movies), and the format of that type (e.g. Microsoft Windows bitmap (BMP), Joint Photographic Experts Group (JPEG), and Audio Video Interleave (AVI)). In Figure 2 is presented a weather related information object that includes a payload. Only the metadata would be processed by a JBI server.

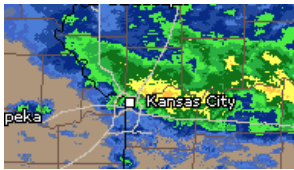
<pre> <metadata> <weather> <temperature> <value>50</value> <scale>F</scale> </temperature> <location> <city>Kansas City</city> <state>Kansas</state> </location> </weather> <filetype>jpg</filetype> </metadata> </pre>
<p>Payload</p> 

Figure 2. Metadata and Payload Information Object

Actual JBI information objects are more tightly defined than the examples above, and the exact specifications are presented in “JBI Detailed Design Description.” All information objects are described by XML Schemas, which define the exact format of the information object metadata. Before the server accepts a publication, the server first validates the requested information object type with the Schema stored in the Meta Data Repository (MDR). If a newly published information object satisfies the stored Schema requirements, then that information object is accepted and processed. If the Schema requirements are not met, then the information object is simply dropped.

2.1.2 Interactions of Information Objects with JBI Servers

There are two ways that information objects interact with JBI servers. The initial interaction occurs when an information object first arrives at the publication service. The publication service retrieves the stored Schema definition and validates the published information object with that Schema definition. Validation is the process of ensuring that the right XML elements are located at the correct relative locations, and that the value types match the ones that were mandated in the Schema definition. This evaluation process is completed on a structural level, and results in a simple true or false statement that states whether or not the information object satisfied the stored definition.

The second type of information object/JBI server interactions occurs during Extensible Metadata Platform (XMP) Path Language (XPATH) predicate evaluations. Subscriber and Query users supply XPATH expressions which specify a set of requirements that define the type of information objects that are of interest. An XPATH expression consists of one or more predicates, and a predicate consists of clauses. A clause is a logical value comparison that results in a true or a false. For example, */metadata/weather/temperature/value < 70* is a clause. A predicate is a logical expression operating on multiple clauses, i.e. */metadata/weather/temperature/value < 70 AND /metadata/weather/temperature/scale = “F”*.

In the above examples, the exact element specified by the XPATH expression is extracted from the XML file, cast into the type defined in the Schema definition, and then compared to the provided static value. Though the information object values are actually

extracted and interpreted by the JBI server, the values are only used for simple comparisons, and are never processed.

2.1.3 Malicious Code Embedded in Information Objects

The contents of the information object are never processed, and malicious code that might be embedded into an information object can never affect the JBI server itself. The only foreseeable avenue of adverse attack through an information objects is by exploiting buffer overflow vulnerabilities in the XML parser that extract element values and casts those elements into the appropriate types. These vulnerabilities are not a problem as long as the latest software updates for the XML parser libraries are applied. Another means of preventing such attacks is by ensuring that all variable length values are limited to a maximum length that is less than the maximum assumed buffer length used in the XML parser libraries.

2.2 JBI Services

There are six types of JBI services. These JBI services are web administration, authentication, metadata repository, publish, subscribe, and query. Each of these services is described below. In practice the JBI should remain operational at all times, which will ensure that users are able to access information and information transportation services when needed without time restrictions.

2.2.1 Web Administration Services

The JBI Web Administration Service is primarily used by JBI system administrators to maintain user accounts and JBI servers. In addition, administrators use this service to make changes to the Information Object Repository (IOR) and the Metadata Repository (MDR). Changes made to the MDR, which are immediately implemented, only affect the JBI server and not the underlying system. The allowable changes are displayed in the form of fill-in text boxes and expanding trees for privilege assignment. All of the available commands are predefined, and are presented in Table 1.

This service allows administrators to log onto the server to add and remove user and information object descriptions. Administrators are also able to view the current server status and statistics and to make changes to the server, such as forcing a server to restart or flushing the repository contents. The following is a list of commands available to the administrator. The *get** commands require only *read* access while the others require *write* access. The resulting changes affect the JBI server, but not the node on which the server is deployed.

Table 1. JBI Web Administration Service Commands

command	purpose
getAccounts()	obtain a list of user accounts
createAccount()	create a new user account
modifyAccount()	make changes to current account
deleteAccount()	delete a current user account
getInfoObjectTypes()	obtain a list of currently supported IO types
createInfoObjectType()	add a new IO type into the JBI server
modifyInfoObjectType()	make changes to an IO type (i.e. update schema)
deleteInfoObjectType()	delete an IO type
getAccountRoles()	obtain Roles that an account hold
setAccountRoles()	give an account new Roles
getInfoObjectCatalog()	obtain a summary of IOs currently in persistent store
archiveInfoObjects()	place IOs in storage, no longer available for query
restoreInfoObjects()	retrive IOs from storage
deleteInfoObjects()	deletes IOs in persistent storage

Administrators can grant clients access to the JBI service from a list of individual hosts, from a community of interest (COI), and/or from other appropriate groupings. Only administrators are allowed to make changes to the server. Administrators can be further separated into individual administrative roles. All users that have been registered with the JBI Web Administrators are allowed service access, and normal users are only able to view data that they have been granted access to. The JBI itself is COI information management system, and so all users would be part of a common COI.

The administrative service uses the standard client and server web architecture. A client is connected through a network to the server which resides on a high performance computer (HPC). Data is transferred in a predefined text format. The data is sent to the Web Administration service through a Hypertext Transfer Protocol Secure (https) POST request method. Data is transferred in a predefined XML format, and is normally sent in plain-text. Secure Shell (SSH) Tunneling provides end-to-end encryption to ensure that the information remains confidential. This service listens on port 11010 and maintains access and connection control for the JBI server. Before any user can use a JBI service, the user must first authenticate with and obtain permission from this service. The authentication service completes both user authentication and authorization functions. User authentication ensures that the client is a valid user on the JBI system. Authorization ensures that the user has the appropriate access rights.

2.2.2 Authentication Service

The JBI uses a role based access control system, which assigns each user one or more roles that reflect the information object types and services that the user will need to access. Before a user can use the Publish, Subscribe, Query and/or MDR services, the user must first authenticate with the Authentication Service (Table 2). After successful authentication, the user is able to connect to the services requested.

The JBI utilizes the Role Based Access Control (RBAC) for fine-grained security policy enforcement. Each information type has a list of roles that can be set for the user. These roles fall into two categories, the administrator and the user. User roles are limited in access, such that only the JBI can be accessed, but the user can not make any changes. Administrators, on the other hand, can make certain changes.

Each information object has the following roles associated with it. A username/password pair is sent to the server for authentication. The information is sent through an http POST over a Secure Socket Layer (SSL) connection. POST is an http request method which submits user data to the identified resource. The data is included in the body of the request.

Table 2. JBI Authentication Service Commands

Role	Privilege	Purpose
publish	user	gives user permission to publish this IO type
subscribe	user	gives user permission to subscribe to this IO type
query	user	gives user permission to query this IO type
MDR	user	gives user read access to the MDR service
MDR	admin	gives user read and write access to the MDR service
admin	admin	gives user the privilege to make any changes to this IO type

Each user can assume multiple roles at the same time. The administrator can mix and match different roles to follow the principle of least privilege, which allows the minimum possible privileges be granted to permit a legitimate action. Because most users will require publish and subscribe services, those users will be assigned publish and subscribe roles associated with the specific information objects that are needed.

2.2.3 Metadata Repository Service

An additional method for making changes is through the Metadata Repository (MDR) Service. Like the Web Administration Service, MDR service users are able to obtain a list of the currently supported *Information Objects* and the corresponding *Schema* definitions. Users are also able to update current *Schema* definitions, and to also add new ones. These changes are forwarded to the Information Management Staff (IMS) for review. Once the changes are approved, the changes are finalized and the MDR is updated.

2.2.4 Publish and Subscribe Services

The Publish and Subscribe services are the most widely used basic services. The Publish service is used to send Information Objects to the JBI server for distribution. The Subscribe service is used by the user to register interest in a specific type of as input/output (IO).

By registering with the Subscribe service, subscribers automatically receive new publications that match the criteria provided to the JBI server. Criteria are presented to the JBI server through predicates in the form of XPATH expressions. Information objects are archived in an Information Object Repository (IOR), which is a persistent database. The JBI server itself does not decide whether or not an IO should be archived. The publisher instructs the server on what it should do during the initial Publish connection sequence.

2.2.5 Query Services

Query is the third basic JBI service, and is similar to Subscribe, except that the archived Information objects are now of interest to the user. When new Information objects are published into the JBI server, the Query users do not receive a copy of the Information objects. However, when the user queries archived information and a query matches stored information, then the payload and the metadata for that IO are sent to the requestor.

3. 100X JBI SECURITY

The JBI Reference Implementation was parallelized in this effort to run more rapidly on high performance computers and Linux clusters. There were additional security concerns that were addressed so that not only could the 100X JBI be implemented on networks, but also so that the new implementation could be accessed interactively.

3.1 Secure Shell Tunneling

The six services that were introduced in Section 2 are the services that users can interact with. The 100X JBI server was deployed on High Performance Computing (HPC) nodes, and the HPC's are behind strict firewalls. Secure Shell (SSH) Tunneling is used to move Information objects through the firewalls. SSH Tunneling provides a secure means for clients to connect to JBI services without having to open up new ports on the firewalls.

SSH Tunneling (2) is a technology that allows packets from a client port to be sent through an SSH connection to a specified server port. One can think of SSH tunneling as a specialized Virtual Private Network (VPN) connection. SSH Tunneling not only provides a means for port communications between a client and a server, but it also provides security features.

Access to the JBI server is restricted to users who have SSH access to the machine where the server is running. This means that for the work presented here access to the JBI server is limited to users who have access to the HPC center where the JBI server is hosted. This feature provides the first level of defense, such that even though a JBI user account might be compromised, the malicious user will not be able to connect to the JBI server without that user first having SSH access to the HPC center.

All of the data transported through the SSH tunnel are encrypted. This is an important attribute since it provides confidentiality. In the current JBI, users authenticate themselves by sending an XML document with their *username/password* in plain text.

The confidentiality that SSH provides ensures that malicious users on the network are not able to simply sniff the user's account information.

All client connections to the JBI rely on the underlying SSH connection. If the SSH connection is broken or has been purposely taken down, client connections will also be forcefully terminated.

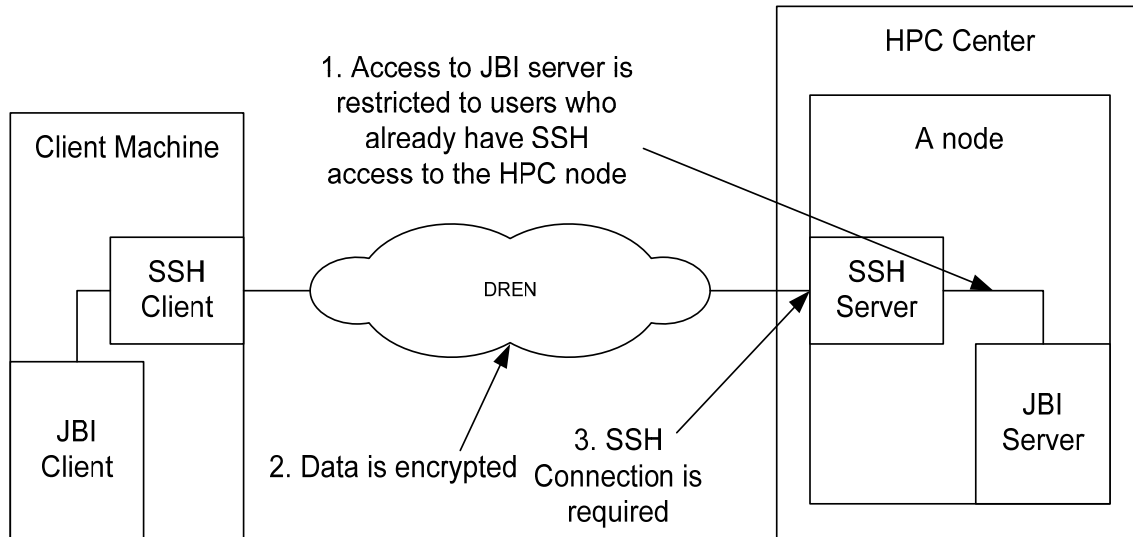


Figure 3. SSH Tunneling

In Figure 3 is presented a more complete overview of SSH tunneling. This service uses the HTTPS internet protocol, which is HTTP over a Secure Sockets Layer (SSL) link, and is configured to listen to client connections on port 8443. The SSH protocol is also used to provide a tunnel between the client and the JBI server.

3.2 Information Objects

The 100X JBI operates on generic Information Objects, and only non-sensitive information was managed in this prototype 100X JBI development. Proper security precautions must be taken before any sensitive material is to be managed by the 100X JBI. The level at which the service is run (user privileges), the level of access required

(access to root or any other non-user accounts, indirect access via setuid, setgid, or other means), and the security level of the network over which the software is installed must be considered.

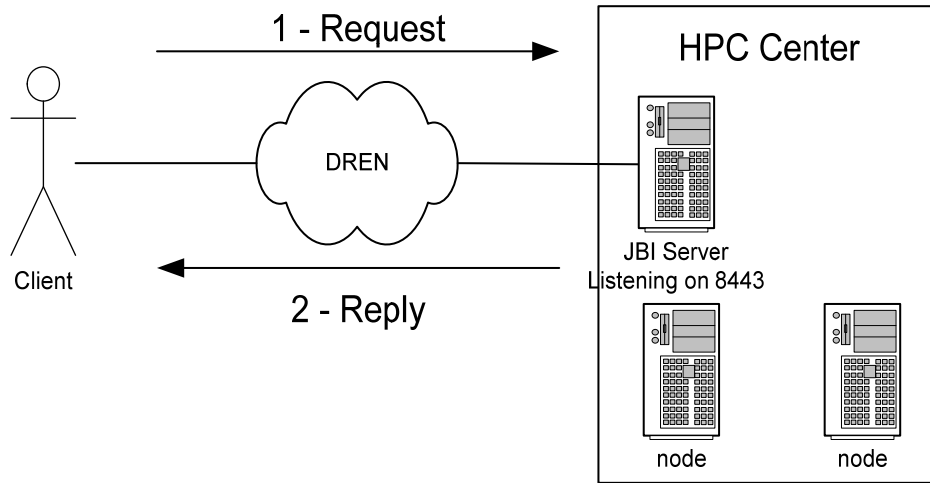


Figure 4. 100X JBI Data Flow Diagram

In Figure 4 is presented a diagram of data flow for the 100X JBI. This is a standard web service that uses the request and reply interaction scheme. The only difference between this service and a normal HTTP web administration service is that normal HTTPS servers listen on port 443 and this service listens on port 8443. Furthermore, access to this service must be preceded by obtaining access to the node on which the service is running. In general this means that the user must obtain a Kerberos ticket and then establish a SSH connection and setup port forwarding.

3.3 Authentication

The Authentication service is central to the JBI, and is used by all the other JBI services to ensure that users are permitted to exercise whatever task they are requesting. Depending on the requested service, authentication takes on a different shape. Publish, Subscribe, Query and MDR users must all first authenticate directly with the

Authentication to obtain a Connection ID. Once a Connection ID is obtained, the user can then connect with the specific service of interest.

Web Administration users authenticate using a less direct method by sending their credentials to the Web Administration service. This service then uses that information to authenticate the administrator with the Authentication service.

3.3.1 Web Administration

A Web Administration user sends authentication information to the Web Administration service, and that service in turn authenticates the user with this Authentication service. Authorization is done on a per-command basis. Whenever the user initiates one of the commands described in Section 2.2.1, this Authentication service is consulted to ensure that the user has the proper permissions to complete the command.

3.3.2 Users

A user in this category authenticates with this service directly. Upon successful authentication, the user will then establish a connection sequence with the specific service he wants to use. The service will take the user's information and consult with this service to ensure that he is authorized to use the requested service.

3.3.3 The Publish, Subscribe, Query and MDR

The publish, subscribe, query and MDR services follow a similar procedure to that of the user for establishing connections with users.

3.3.4 Kerberos

Firewalls, while offering security from deleterious and/or malicious effects from outside a network, restrict the flexibility of an internet based network. To use the 100X JBI as an information management system in the field, the distributed users would necessarily be outside the firewall. Because the 100X JBI was developed to run on the Distributed Interactive High Performance Computing Testbed (DIHT), which is based upon the Defense Research and Engineering Network (DREN), there are firewalls rules

imposed by the High Performance Computing Modernization Program (HPCMP) that must be followed. To use any of the 100X JBI services, the user must first be authenticated to access the assets of the DIHT.

Kerberos is a network authentication protocol, and was designed to provide strong authentication for client/server applications by using secret-key cryptography. Users must be running Kerberos software on the local computer and have a one-time password SecurID card issued by the DoD's High Performance Computing Modernization Program (<https://kirby.hpcmp.hpc.mil/>). The SecureID utilizes a separate password or Personal Identification Number (PIN) and an authenticator (<http://www.rsasecurity.com/node.asp?id=1156>). The SecureID card supplements the password based authentication mechanism by adding the additional requirement of a passcode derived from a time-dependant code generated by the SecureID card. This tests something you know and something you have, and are two categories or classes of authentication mechanisms.

Kerberos is a two stage process. First the user authenticates with the Authentication Service, which is run on a Kerberos Server. The server sends the user a Ticket Granting Ticket (TGT). When the user wants access to a computer on the DIHT, the user sends a TGT to a Ticket Granting Service (TGS), and the TGS returns a Session Service Ticket (SST). The user then uses the SST to authenticate and access the DIHT computer. The Authentication Service and the Ticket Granting Service are two different entities, and can be hosted on the same server.

If more than one computer is to be used on the DIHT, then SST's for each computer must be obtained. An issued SST has a set period of time before it expires. In cases where extended use is required, either new tickets will need to be obtained, or arrangements need to be made for longer duration tickets.

4.0 100X JBI ARCHITECTURE

The architecture of the 100X JBI project is based on the AFRL JBI Reference Implementation 1.2. The JBI Reference Implementation architecture adopted software methodologies that provided maximum flexibility for experimenting with new features, and was independent of performance. The conceptual framework for a net-centric pub/sub system included a standard Common Application Programming Interface (CAPI). This Reference Implementation was based on a proof of concept that has been implemented to interoperate in a net-centric system environment.

The 100X JBI goals included a two-order of magnitude speedup over the reference implementation. To achieve this goal, a two fold approach was implemented, which focused on converting JAVA codes to C++, and then parallelizing the codes. The architecture of the 100X JBI is based upon the JBI 1.2 Reference Implementation architecture, which is presented in Figure 5.

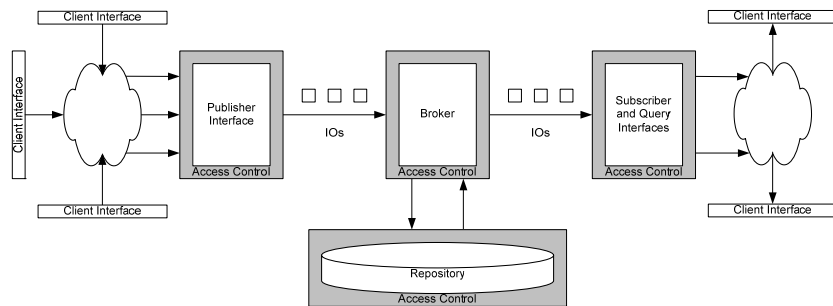


Figure 5. Overview of the 100X JBI architecture.

Information objects arrive at the publisher interface, are sent to the broker which sends the publication to repository, and/or matches the publication against earlier subscriptions. When a match is found, the information object is forwarded to the subscriber who requested it. If a query arrives at the publisher interface, that query is compared to the stored publications in the repository, and if a match is found, the resultant publication is forwarded to the requestor.

4.1 Communications

The first step in the implementation of the 100X JBI was to speed up the basic communications. The configuration of the JBI reference implementation was changed to bypass the Java Messaging Services (JMS) layer, and the Java JNI (JavaTM Native Interface) communications software was replaced with a TCP/IP socket implementation in C++. The communications changes affected the JBI core services implementation as well as the CAPI library interface.

The `JniConnection` and `JniConnectionService` were the first Java components of the reference implementation that were replaced in the 100X JBI. The initial change in communications software made a significant improvement in speedup performance. As expected, the C++ compiled code performed faster than the Java Virtual Machine-based system. These initial results were encouraging, and language conversion from Java to C++ was one of the principal approaches used for 100X JBI performance enhancements.

`JniConnection` is the Java Native Interface (JNI) wrapper to call the C++ JBI client libraries from Java. `JniConnectionService` is the Java Native Interface C++ code that enabled 100X JBI clients in C++ or Java to communicate to the original JBI Java server code that provided the security framework.

4.2 Publishing

The publication service was the next piece of the JBI that was considered for high performance implementation. Initially, publisher sequence support was added to the C++ JNI communications interface, and a publication catcher component was added to the 100X JBI server. The CAPI was tested to validate the connections to remote JBI servers. The publication catcher received publications from CAPI clients and forwarded the publications to the brokers with which it was configured to interoperate.

Publication services required Extensible Markup Language (XML) libraries to be added to the implementation along with some other basic JBI utilities for constructing messages and for control over exceptions. Initial testing of the Field Programmable Gate Array (FPGA)-based XML filtering support was conducted by filtering messages passing through the pubcatcher, although the broker had not yet been implemented.

4.3 Subscribing

Subscription services were added to the 100X JBI. The changes involved the JNI implementation, the core services and the CAPI. Subscriber services included registering subscriptions with a broker and establishing “object available callbacks” to receive notification when brokers discovered publications that match the subscriptions.

4.4 Brokering/Disseminating

Brokering services match publications with interested subscribers, and metadata comparisons determine the desirability of publications for subscribers. Registered subscriptions that match a received publication are forwarded to the user through the object available callback subscriber method. The broker uses a disseminator process to distribute publications to users, which also avoids delays in sending requested publications over possibly slow network connections. The broker used shared memory to enhance its performance.

4.5 JBI Connectors

The parallel distributed high performance 100X JBI used “JBI Connectors” to handle distributed operations. In a 100X JBI information management system, there may be several JBI servers, each implementing publication and subscription services. 100X JBI connectors forward messages to remote JBI’s, where a subscription may be brokered. The Message Passing Interface (MPI) was used to improve the performance and allow 100X JBI server scaling for high performance computer implementations. Peer services allowed publication catchers to forward publications to remote 100X JBI servers. This architecture allowed multiple publication catchers for each JBI.

4.6 Query and Archive

Publications can be available for users who are not connected when a publication of interest is published. Users can request archival services when publishing. A query function allows users to request archived documents using a syntax that was similar to the subscription syntax. Along with the archival and query services, the ability to handle payloads and, for enhanced performance, large memory-based payloads with metadata were added to the 100X JBI.

4.7 100X JBI Architecture

In Figure 6 is presented a more detailed overview of the 100X JBI architecture. At the top of the figure is shown that the authorization credentials for publish requests and subscription requests are sent to the Connector Manager, which then connects to the Information Peer List. A more complete description of this process is presented in Section 5.2.

When incoming publications are received from other HPC’s and/or from authorized clients, these publications are temporarily held in the Publisher Catcher. There can be from one to m Publisher Catchers. When there is more than one Publisher Catcher, then each Publisher Catcher is located on a separate processor. The publications in the Publisher Catchers are sent to a Connector (a single processor), which then transmits a replicate of the publication to from zero to three other HPC’s. If other HPC’s

are being transmitted to, then the system has redundancy. The publication is also sent to the next available Broker.

There can be from one to n numbers of Brokers, and if more than one Broker is being used, then each Broker is on a separate processor. A Broker compares the elements of the metadata with the predicates for each subscription that has been previously submitted. When a match is found, then the corresponding publication is sent by the next available Disseminator (one to k total Disseminators), which then transmits the publication to the client that had submitted the subscription.

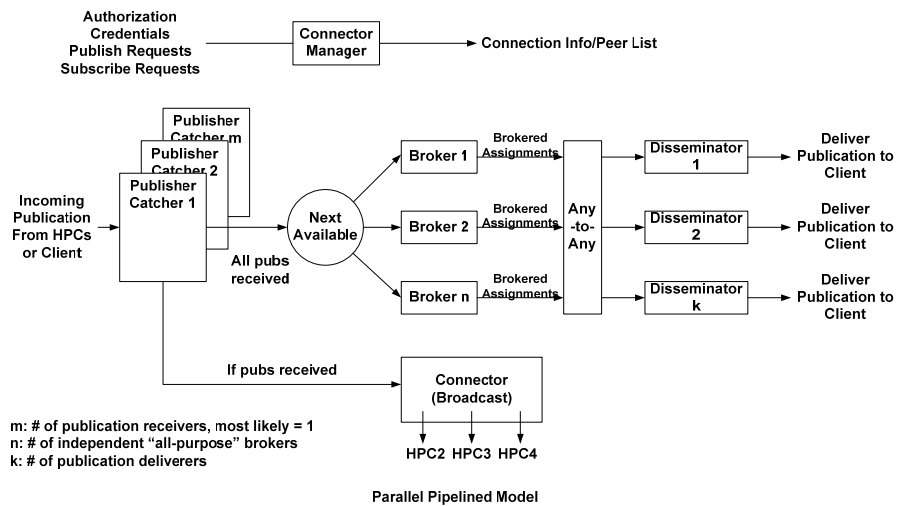


Figure 6. Detailed Overview of the 100X JBI architecture

5. 100X JBI System Sequence Diagrams

In this chapter Unified Modeling Language (UML) System Sequence Diagrams (SSD's) are presented for the 100X JBI architecture. For each use case scenario, the events that the user generates, the order that those events are generated in, and the responses of the system to the generated event are listed.

5.1 Publication

In Figure 7 is presented the SSD diagram for the 100X JBI publication service. After the Kerberos ticket is issued to the publisher, an SSH connection is requested through Port 22 of the high performance computer or Linux cluster on which the JBI server is located. When the connection is granted, a SSH tunnel is created from port 11010 on the local host to Port 11010 on the HPC. Authentication with the JBI server is achieved through this tunnel. A unique Connection ID Number is issued by the JBI server and transmitted back to the publisher through SSH tunneling.

The setup of the publication sequence then occurs, in which the Connection ID Number, notification that this is a publication, the IO type, and the IOVer are SSH tunneled to Port 11011 of the JBI server. The JBI server then issues a unique Publication Sequence ID Number to the publisher by SSH tunneling.

The publisher then uses this unique Publication Sequence ID Number in future communications with the server. If the publisher chooses to publish an information object, the whole object is serialized and sent through the SSH tunnel to the server along with the Sequence ID Number. Once the server acknowledges that the publication has met the schema standards, a unique Object ID is sent back to the publisher. In the case when acknowledgements are not request by the publisher, a unique ID is not sent back to the publisher.

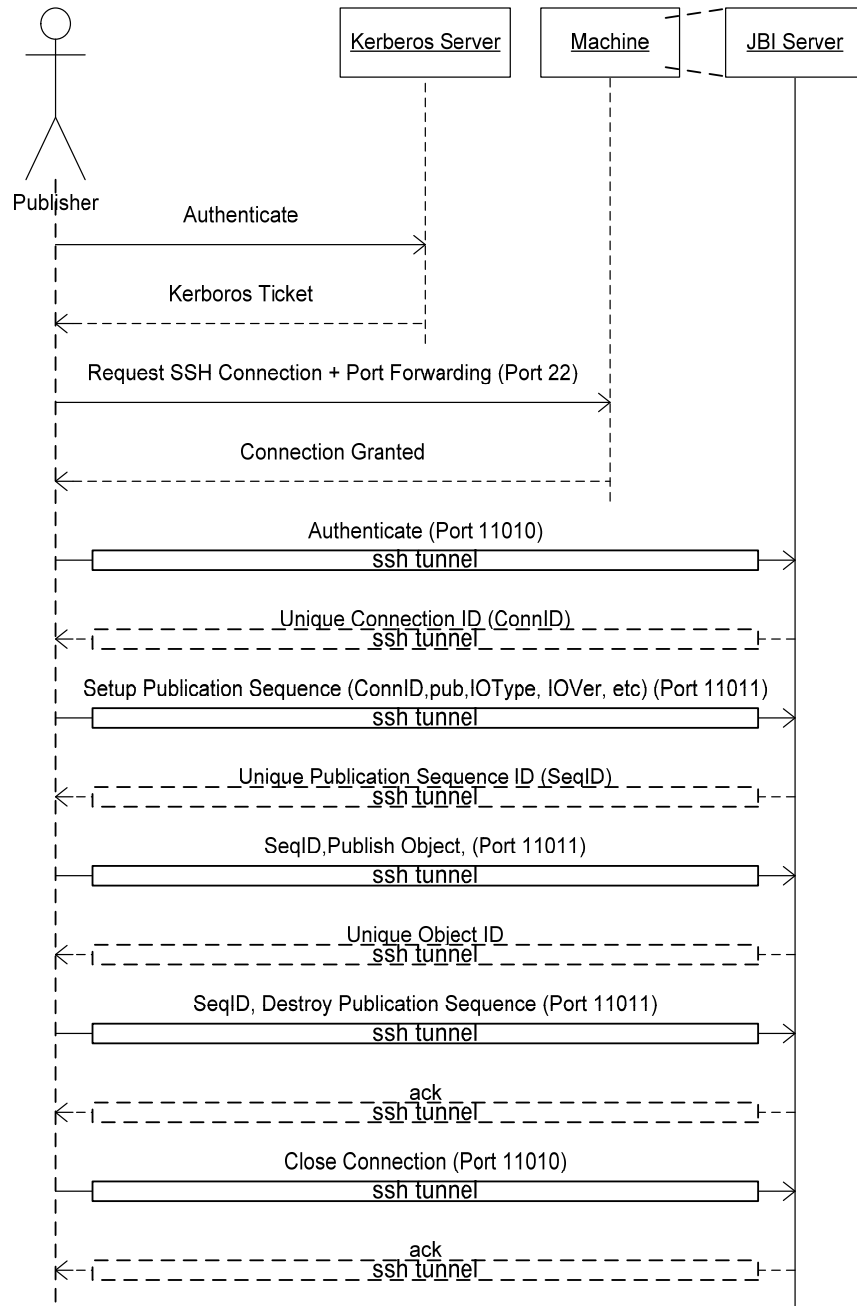


Figure 7. Publish System Sequence Diagram

5.2 Subscribe

In Figure 8 is presented the SSD for the 100X JBI subscribe service. After the Kerberos ticket is issued to the subscriber, an SSH connection is requested through Port 22 of the high performance computer or Linux cluster on which the JBI server is located. When the connection is granted, an SSH tunnel is created from port 11010 on the local host to Port 11010 on the HPC. Authentication with the JBI server is achieved through this tunnel. A unique Connection ID Number is issued by the JBI server and transmitted back to the subscriber through SSH tunneling.

The setup of the subscription sequence then occurs, in which the Connection ID Number, notification that this is a subscription, the IO type, the IOVersion, and the XPATH predicate are SSH tunneled to Port 11012 of the JBI server. The JBI server then registers the predicate for future matching of incoming publications. The predicate is stored in a lookup table that any broker can access.

Should a publication arrive at the JBI server which matches the predicate data of the subscription, then the sequence ID of the matching publication is delivered to the subscriber by SSH tunneling by the JBI server. The subscriber immediately receives the IO if the IO is less than 128x8 in size. If the size of the IO is large than this, then the subscriber must send an acknowledgement back to the server before the server sends the IO to the subscriber. Once the subscription is completed, the subscriber requests to the JBI server that the subscription sequence be destroyed through Port 11012. The JBI server acknowledges the request to the subscriber, and the subscriber SSH tunnels to the JBI server that Ports 11010 and 11012 are to be closed. Through SSH tunneling the JBI sever acknowledges to the subscriber that the ports are closed, and the subscription sequence is completed.

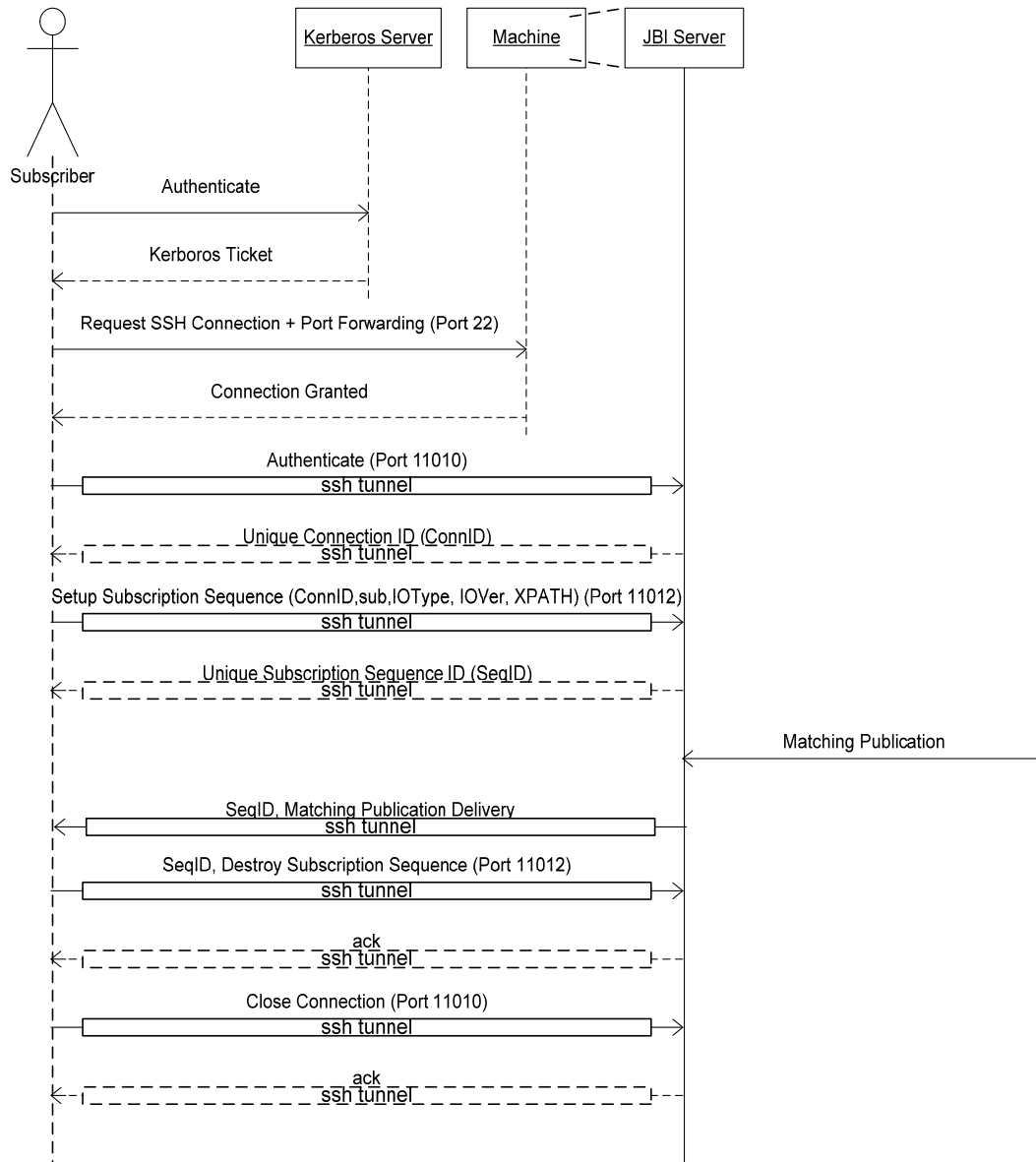


Figure 8. Subscribe System Sequence Diagram

5.3 Query

In Figure 9 is presented the SSD for the 100X JBI query service. The query user must first authenticate to request a Kerberos ticket from the Kerberos Server. When the Kerberos ticket is issued to the query user, an SSH connection is requested through Port 22 of the high performance computer or Linux cluster on which the JBI server is located. When the connection is granted, then SSH tunneling authentication is achieved from the JBI server through Port 11010. A unique Connection ID number is issued by the JBI server to the subscriber through SSH tunneling.

Next the setup of the query sequence occurs, in which the connection ID number, notification that this is a query, the IO type, the IOVER, and the XPATH are SSH tunneled to Port 11013 of the JBI server. The JBI server then issues a Unique Sequence ID Number for the query by SSH tunneling to the query user. The JBI server then checks the JBI repository for matching predicates.

A list of matching information objects is sent back to the query user and the query user determines which resulting information objects he wants.

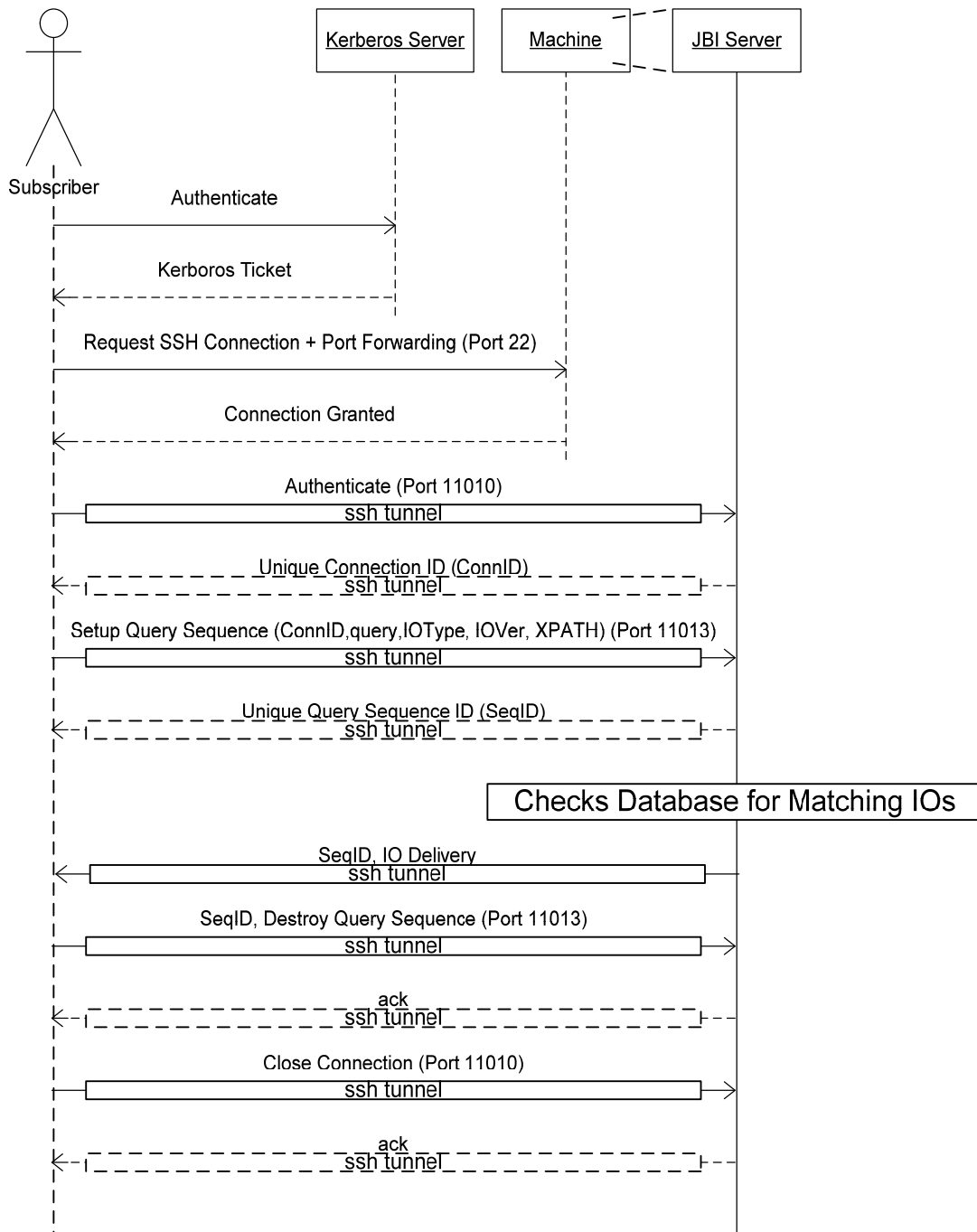


Figure 9. Query System Sequence Diagram

5.4 Meta Data Repository

In Figure 10 is presented the SSD for the 100X JBI Meta Data Repository (MDR) service. The user of the MDR must first authenticate to request a Kerberos ticket from the Kerberos Server. When the Kerberos ticket is issued to the user, an SSH connection is requested through Port 22 of the high performance computer or Linux cluster on which the JBI server is located. When the connection is granted, then SSH tunneling authentication is achieved from the JBI server through Port 11010. A unique Connection ID number is issued by the JBI server to the user through SSH tunneling.

The JBI server then searches for a matching Schema. If a matching Schema is found, then the user is notified that a matching schema was located by SSH tunneling. The user then makes a Schema request over Port 11014 by SSH tunneling, and the JBI server SSH tunnels back the SCHEMA. The user then sends to the JBI server over Port 11014 by SSH tunneling a request to destroy the MDR sequence request, and the JBI server by SSH tunneling replies that the MDR sequence request has been destroyed. Next the setup of the MDR sequence occurs, in which the connection ID number, notification that this is an MDR, the IO type, the IOVer, and the Update are SSH tunneled to Port 11014 of the JBI server. The JBI server then sends a unique MDR Sequence ID number by SSH tunneling to the user. The user then sends to the JBI server the new Schema by Port 11014 by SSH tunneling, and the JBI server acknowledges back to the user by SSH tunneling the receipt of the Schema request. The administrator of the JBI server is notified that a new schema has been submitted for review. The user then SSH tunnels the JBI server to destroy the MDR Sequence ID over Port 11014, and the JBI server acknowledges to the user by SSH tunneling that it was destroyed. The user then requests by SSH tunneling that the connection be closed through Port 11010, and the JBI server acknowledges by SSH tunneling that the connection has been closed.

Should a publication arrive at the JBI server which matches the predicate data of the subscription, then the sequence ID of the matching publication is delivered to the subscriber by SSH tunneling by the JBI server. The subscriber then requests to the JBI server that the subscription sequence be destroyed through Port 11012. The JBI server acknowledges the request to the subscriber, and the subscriber SSH tunnels to the JBI

server that Port 11010 is to be closed. Through SSH tunneling the JBI sever acknowledges to the subscriber that the port is closed, and the subscription sequence is completed.

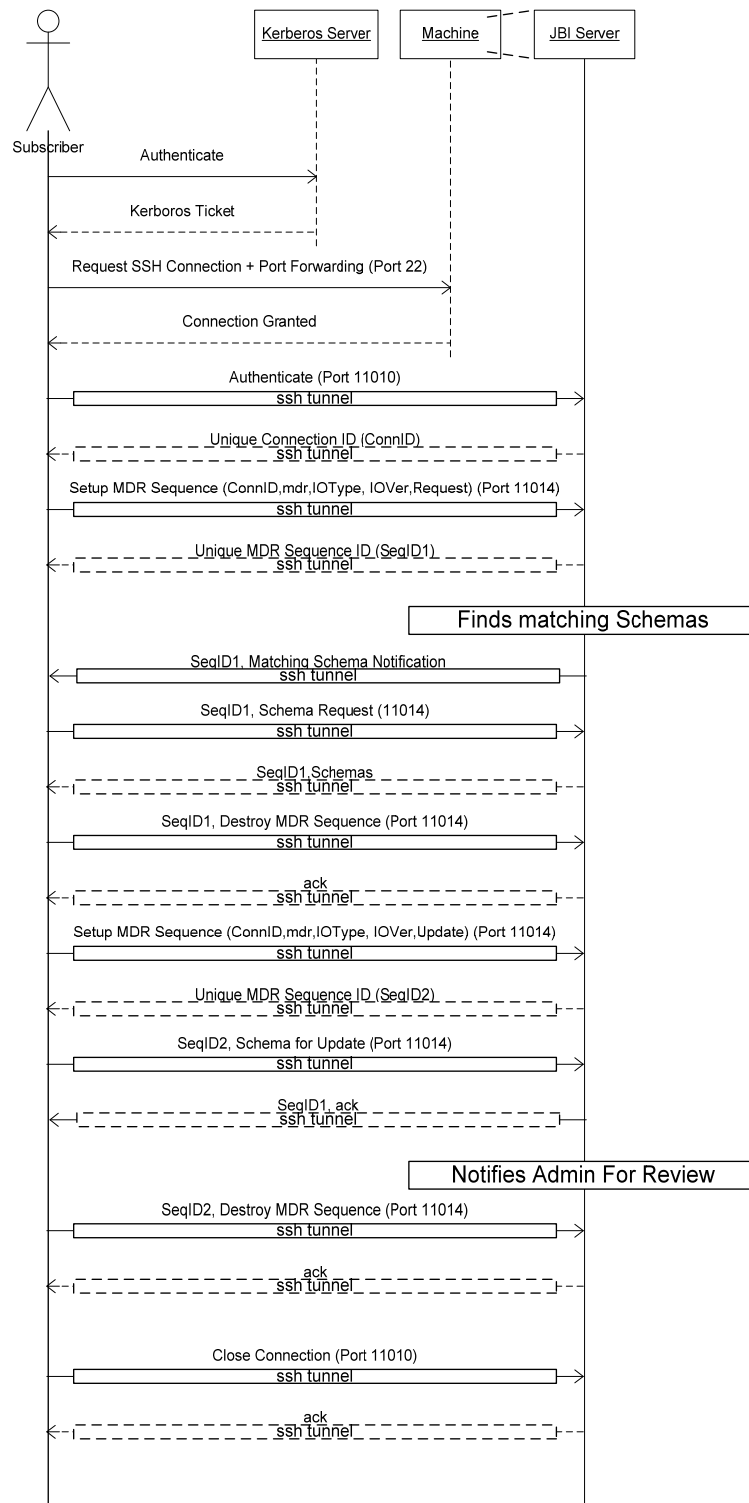


Figure 10. Meta Data Repository System Sequence Diagram

6. DISTRIBUTED INTERACTIVE HPC SYSTEM

6.1 Overview

The Distributed Interactive HPC Testbed (DIHT) is an experimental testbed that is hosted on the Defense Research and Engineering Network (DREN) (<http://www.hpcmp.hpc.mil/Htdocs/DREN/index.html>), and was recently implemented by the Information Directorate of the Air Force Research Laboratory to provide scientists and engineers with capabilities for high performance computing that are distributed over a wide geographic area with real-time interactive responsiveness. The mission of the DoD's High Performance Computing Modernization Program (HPCMP), which sponsors the DREN, is to develop high performance computing (HPC) capabilities within the DoD's Research, Development, Test & Evaluation (RDT&E) community.

The HPCMP accomplishes its mission by providing access to HPC machines to the DoD community through three categories of sites. The Major Shared Resource Centers (MSRC's) house large supercomputers and provide computing cycles to users across the nation. The Distributed Centers (DC's) deploy more modest systems, satisfy more local needs, and enable the host organizations to stay at the forefront of HPC technology. The DC's strive to develop new software applications and/or evaluate advanced computing and communications technologies. There are also DoD User Sites that are geographically dispersed on the DREN. The DIHT is a collaboration of one MSRC, one DC and 2 User Sites that provide interactive and distributed capabilities. Additionally, other distributed authorized users of the DIHT need only connect to the DREN to gain access to the testbed.

6.2 Linux Clusters

In Figure 11 is presented the locations of the distributed Linux clusters that are integrated together on the DIHT, and a more complete description of the clusters is presented in Table 3.

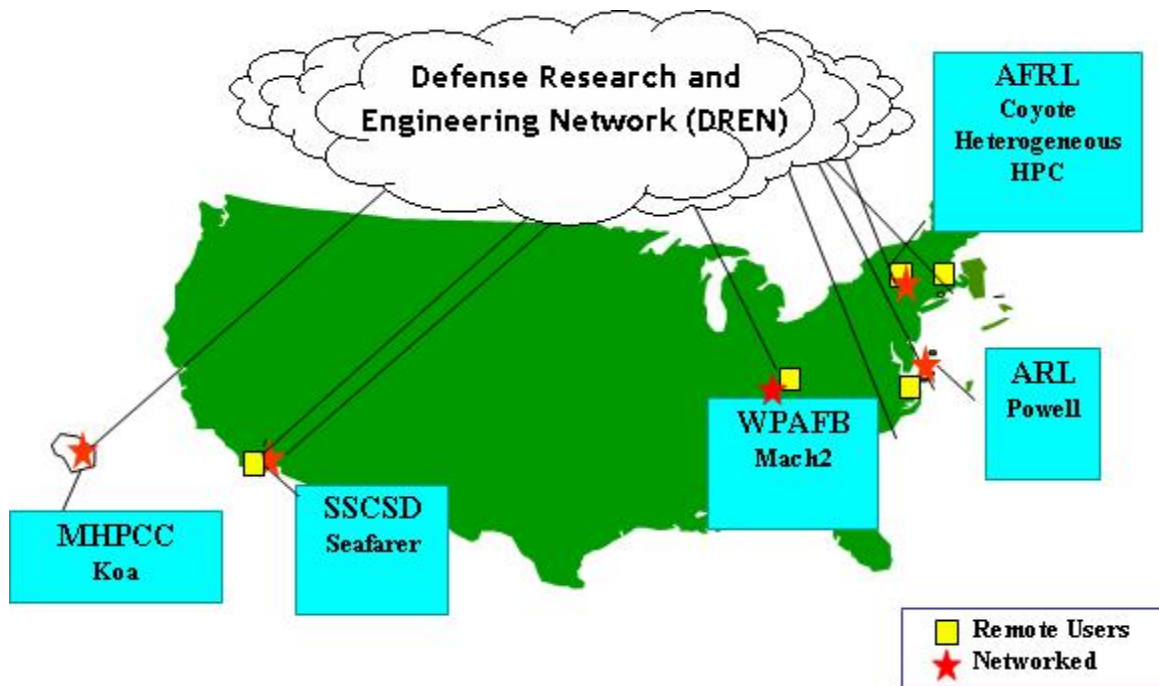


Figure 11. Distributed Interactive HPC Testbed

Table 3. DIHT Linux Clusters

Linux Computer Cluster	Location	Processors	Memory	I/O
Powell	ARL MSRC Aberdeen, MD	128 node dual 3.06MHz Xeon	2 GB DRAM 64 GB disk/node	Myrinet & GigEnet 100MB Backplane
Mach2	ASC MSRC Dayton, OH	24 node dual 2.66 GHz Xeon	4 GB DRAM 80 GB disk/node	Dual GigEnet
Coyote	AFRL Rome, NY	26 node dual 3.06 GHz Xeon	6 GB DRAM 400 GB disk/node	Dual GigEnet
HHPC	AFRL Rome, NY	48 node dual 2.6 GHz Xeon Wildstar II FPGA cards	2 GB DRAM 64 GB disk/node	2Gb Myrinet & GigEnet
Seafarer	SSCSD San Diego, CA	24 node dual 3.06 GHz Xeon	4 GB DRAM 80 GB disk/node	Dual GigEnet
Koa	MHPCC Maui, HI	128 node dual Xeon	4 GB DRAM 80 GB disk/node Shared file system	Dual GigEnet

6.2.1 Mach 2

At the Aeronautical Systems Center (ASC) MSCR is the Linux cluster Mach2, which is a 24 node, 2.66 GHz dual Intel Xeon cluster with 4 GB Dynamic Random Access Memory (DRAM) and 80 GB disk per node and dual gigabit Ethernet (GigE) interconnection fabric. This system uses Red Hat Linux Enterprise 3 as its operating system.

6.2.2 Powell

At the Army Research Laboratory (ARL) MSCR is located the Linux Cluster Powell, which is a 128 node, 3.06 GHz dual Intel Xeon cluster with 2 GB DRAM and 64 GB disk per node with Myrinet and GigE interconnection fabrics and a 100 MB backplane. Powell is comprised of four different types of nodes: compute, storage, login, and management. All nodes contain dual Intel XEON processors integrated with an Intel 7501 chipset, 512k of Level 2 cache and 2 GB of ECC-protected memory. The compute and login nodes have 3.06 GHz processors while all others have 2.4 GHz processors. The 128 compute nodes total 256 processors and 256 GB of memory with a peak system performance of 1.566 Teraflops. All compute nodes have a Myrinet 2000 interconnect which offers 2 Gbps bandwidth into and out of each node. The interconnect is non-blocking and supports MPI communications. The latency on the switch is less than 6 μ s or 250 MB/sec bi-sectional bandwidth.

The storage nodes provide the cluster with an aggregate of 10 TB disk space, divided among 5 file systems that are globally accessible and offer scalable parallel performance via Sistina's Global File System (GFS). All of the nodes share a gigabit Ethernet network which provides a high-performance communications conduit between the storage nodes and the compute and login nodes. This network also connects to the ARL MSCR backbone network, providing a high-performance path to the mass storage archival system. In addition, all nodes have access to the Defense Research and Engineering Network (DREN).

6.2.3 Koa

The 128 node, dual 3.06 GHz Intel Xenon Linux cluster called Koa is located at the Maui High Performance Computing Center. This cluster has 4 GB of memory per node, and the nodes are interconnected via gigabit Ethernet.

6.2.4 Coyote

At the Information Directorate at the Air Force Research Laboratory is the Coyote Linux Cluster, a 26 node dual 3.06 GHz Intel Xeon cluster with 4 GB DRAM and 400 GB disk per node and a gigabit Ethernet (GigEnet) interconnection fabric.

6.2.5 Heterogeneous HPC

Also at the Information Directorate is the Heterogeneous HPC, a 48 node dual 2.6 GHz Intel Wildstar II Field Programmable Gate Array (FPGA) cluster with 2 GB DRAM and 64 GB disk per node and 2 Gb Myrinet and GigEnet interconnection fabric.

6.2.6 Seafarer

At the Space and Naval Warfare Systems Center, San Diego (SSCSD) is the Linux cluster Seafarer, which is identical to Mach2.

6.3 Testbed Applications

Among the early interactive applications of this testbed has been the testing of the 100X JBI information management system. Typical usage of the HPCMP's HPC resources is via batch mode - where users request some amount of processing time on an HPC resource, submit their jobs to the resource's batch queue, and wait for their jobs to reach the top of the queue for execution. This could take several hours or even several days - depending upon many factors. There exists a need within the DoD HPC user community for an interactive capability in which the user requires the result within minutes or even seconds.

One prominent justification for pursuing distributed computing (also known as grid computing, networked computing, or meta-computing) is to leverage computer resources that are perhaps tens to hundreds of times more powerful than is typically housed at a single facility. But there are also two other, equally strong justifications – namely: (1) In typical battlespace environments, the data (e.g., sensors) are inherently geographically dispersed; hence distributing the computing resources close to the data saves the bandwidth and latency required to communicate them to a centralized processor. And (2) the “players in the game” (i.e., the warfighters) are inherently dispersed hence having multi-source data-fusion and battle-planning processors close to the local decision makers makes very good sense.

It should be noted that this testbed is inherently well suited to explore paradigms for network-centric warfare (whose requirements are inherently highly distributed and interactive). It is expected that experimental results will benefit the Air Force’s C2 Constellation and Joint Battlespace Infosphere programs, the Navy’s FORCEnet program, and the Army’s Future Combat Systems program.

7.0 CORE SERVICE EXPERIMENTATION

Several experiments were designed and run to determine the capacity and speed of the 100X JBI architecture, and the results of these determinations were compared to the JBI Reference Implementation 1.2. The capacities of publisher catchers and brokers to process Information Objects were determined, and then the capacity of the overall 100X JBI to process and disseminate information objects were determined as a function of the complexities of predicate complexities.

7.1 Publisher Catcher Capacity

The publisher catcher receives the incoming information objects from clients and/or other hpc's, and holds those publications in a queue until the next broker is available, at which time that publication is sent to that available broker for processing (Figure 12). The publisher catcher sends both the metadata and the payload to the next broker when the total size of the incoming publication is less than 128 kbytes. If the total size is greater than 128 kbytes, then the payload is sent to memory, and only the meta data is sent to the broker. As a broker processes an Information Object, whenever there is a match, the Information Object is sent to the disseminator, which forwards that Information Object to the requestor.

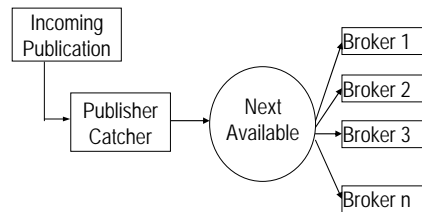


Figure 12. 100X JBI Architecture Publisher Catcher Capacity

7.1.1 Experiment

This experiment was designed to determine the number of information objects that can be processed per time unit by a single Publisher Catcher. It was conducted on the Heterogeneous High Performance Computer, and the information object sizes were 2 kbytes each. Additional Publisher Catcher experimental parameters are presented in Table 4, and other more specific experimental details are presented in Table 5.

In this experiment incoming information objects were submitted to one Publisher Catcher. The information object was then sent through a queue to the next available Broker. The experiment consisted of several iterations in which the number of brokers was increased from 1 to 20, and the numbers of information objects processed per second were recorded. For this experiment each broker was on a separate processor.

Table 4. Publisher Catcher Parameters

Computer	Heterogeneous High Performance Computer (HHPC)
Processor speed	2.6 GHz
Publication Size	2 kbytes
Publishing Processors	1
Broker Processors, n	1, 2, 4, 8, 16, 20
Disseminator Processors	1

Table 5. Publisher Catcher Experimental Conditions

#Run	#brokers	#pub catchers	#nodes
1	1	1	1
2	2	1	2
3	4	1	3
4	8	1	5
5	16	1	9

7.1.2 Results

The results of the experiment to determine the Publisher Catcher capacity are presented in Figure 13. With a one processor brokering system (Run 1), 8,200 information objects of 2 kbyte size were processed in one second. Increasing the number of brokers to two (Run 2) increased the number of information objects being processed to 12,200 per second. Marginal increases to 13,000 information objects per second were realized by increasing the number of brokering processors beyond two (Runs 3-6). From the experimental results presented in Figure 13, the Publisher Catcher capacity was determined to be 13,000 information objects per second, which corresponded with the processing of 26 Mbytes per second.

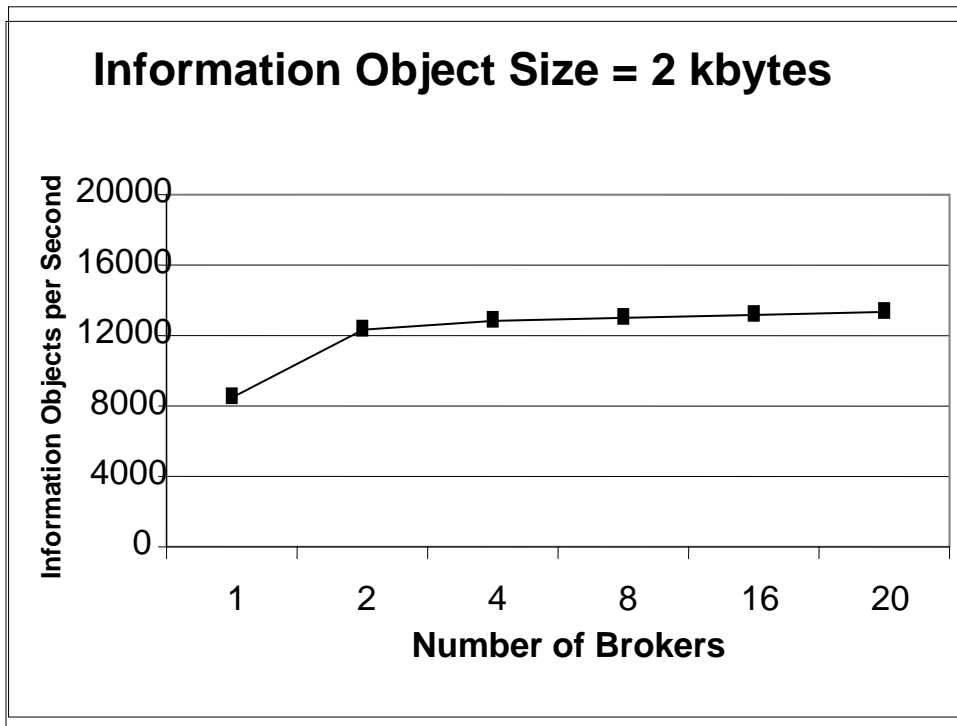


Figure 13. Information Objects Processed v. Number of Brokers

7.2 End to End Single Clause Latency

The input to output latency of the 100X JBI was determined from the amount of time necessary to process an information object. Of interest for this experiment was the determination of the time required to completely process an incoming information object with a predicate that consisted of a single clause on one processor. The number of predicates (subscribers) was increased for iteration during the experiment, and the time to completely process each iteration was measured.

7.2.1 Experiment

In Table 6 is presented the experimental parameters for the determination of the end to end latency for a single clause for the 100X JBI on a single processor. The specific 100X JBI architecture for this experiment is presented in Figure 14. This experiment was conducted on the Coyote Computer, and both the 100X JBI Implementation and the JBI 1.2 Reference Implementation were run on one processor, respectively. The size of the information objects was 1.3 kbytes.

Table 6. End to End Latency Parameters

Computer	Coyote
Processor speed	3.06 GHz
Publication Size	1.3 kbytes
Processors	1
Subscribers	1 to 184

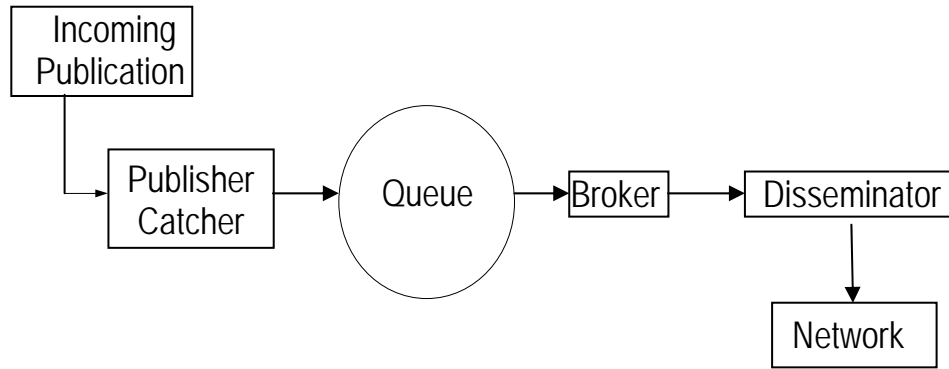


Figure 14. 100X JBI Architecture for end to end latency per predicate with a single processor

This experiment was set up so that one client published a single information object to the JBI server. The number of subscribers was incrementally increased from 4 to 184. The same clause, the trivial XPATH expression (`/metadata/info/size>0`), was used by each of the predicates (subscribers). When the Broker received the information object, it evaluated the clause and determined which subscribers requested the information object. Since all of the subscribers had requested it, the information object was then disseminated to all the subscribers. The end to end latency for this experiment was the time from when the information object was first submitted to the publisher catcher to when the last information object was disseminated to the last subscriber.

7.2.2 Results

In Figure 15 is presented a plot of the determined latency (ms) verses the number of predicates (subscribers) for both the 1.2 JBI Reference Implementation and the 100X JBI. Linear regression analyses have been performed on the data, and the calculated best fitting straight lines are presented in the figure. The slope of the linear least squares fit represented the increase in end to end latency per time unit incurred for additional subscribers. Given this, the data showed that each subscriber added about 2.1 ms of latency, on average, to the system for the JBI 1.2 Reference Implementation and 100 μ s

for the 100X JBI. The 100X JBI resulted in a 21 times improvement in incurred latency when compared with the JBI 1.2 Reference Implementation. With 184 subscribers the latency was 420 ms for the JBI 1.2 Reference Implementation, and 19 ms for the 100X JBI implementation. Because both of these implementations were run on a single processor, the improvement in the latency of the 100X JBI implementation was attributed mainly to the faster execution of the C and C++ codes of the 100X v. the speed of JAVA executions of the Reference Implementation.

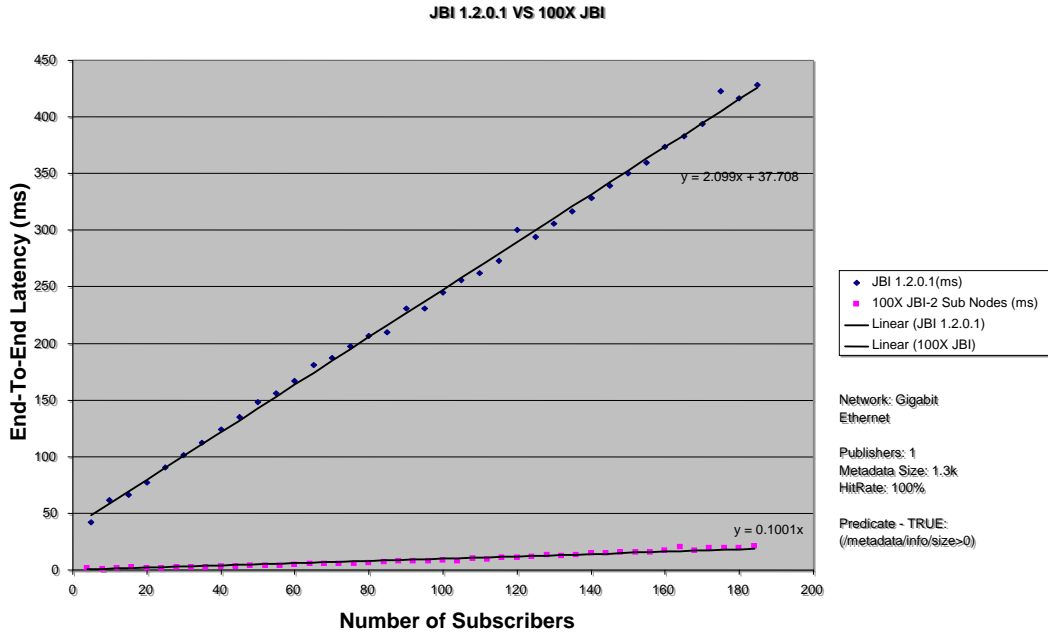


Figure 15. End to end latency v. the number of subscribers (predicate clauses)

7.3 Predicate Complexity

While the previous experiment focused on determining the increase in latency observed by adding more subscribers (predicates), the following experiment focused on the determination of how the complexity of each subscriber's XPATH expression affected the 100X JBI implementation latency.

7.3.1 Experiment

By definition, each XPATH expression is a predicate, each predicate is a conjunction of clauses, and each clause is a comparison test between an element or an attribute of the XML metadata and a value. For example, the simple predicate used in the end-to-end latency test (`/metadata/info/size>0`) consisted of only one clause. An example of a predicate with two clauses is (`/metadata/info/size>0` and `/metadata/info/size<2000`).

For this experiment the predicate complexity was defined as a variable of the number of clauses within each predicate. The experimental parameters are presented in Table 7. This experiment was configured with a single publisher publishing 1.3 kbyte information objects, and with ten subscribers. In this experiment the complexity of the predicates was changed. The architecture of this experiment is presented in Figure 16.

Table 7. Broker Latency Complexity Parameters

Computer	Coyote
Processor speed	3.06 GHz
Publication Size	1.3 kbytes
Publishing Processors	1
Broker Processors	1
Number of Subscribers	10

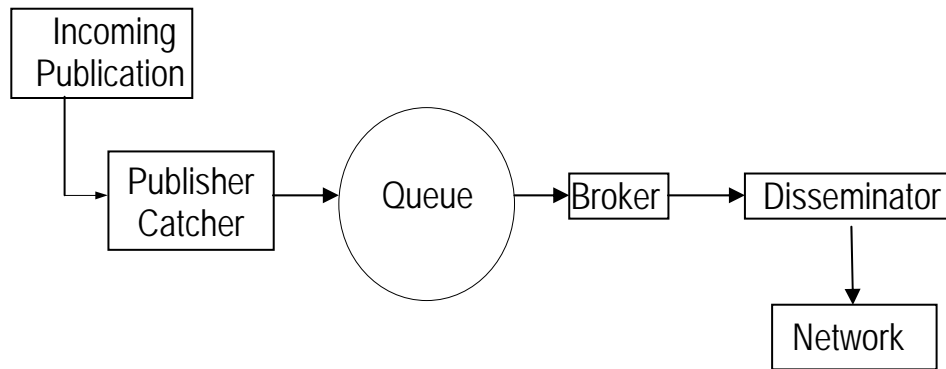


Figure 16. Architecture for end to end latency per clause with a single processor.

Short circuiting is defined as when a system only evaluates as many clauses as necessary to arrive at a final answer. For example, if an "or" was used as the conjunction for two clauses and the first clause resulted in a true, it is unnecessary to evaluate the second clause. Short circuiting would be deleterious for this experiment, and so only the final comparison was true. Because short circuiting was prevented, this experiment isolated the latency effects of one broker evaluating XML documents.

7.3.2 Results

In Figure 17 is presented a plot of the determined latency verses the number of clauses per subscriber. A linear regression has been performed on these results, and the calculated best fitting straight line is also presented in the figure. From the slope of the line, a 10 clauses increase, (1 clause per subscriber for 10 subscribers) resulted in 41 μ s of additional latency for the 100X JBI implementation.

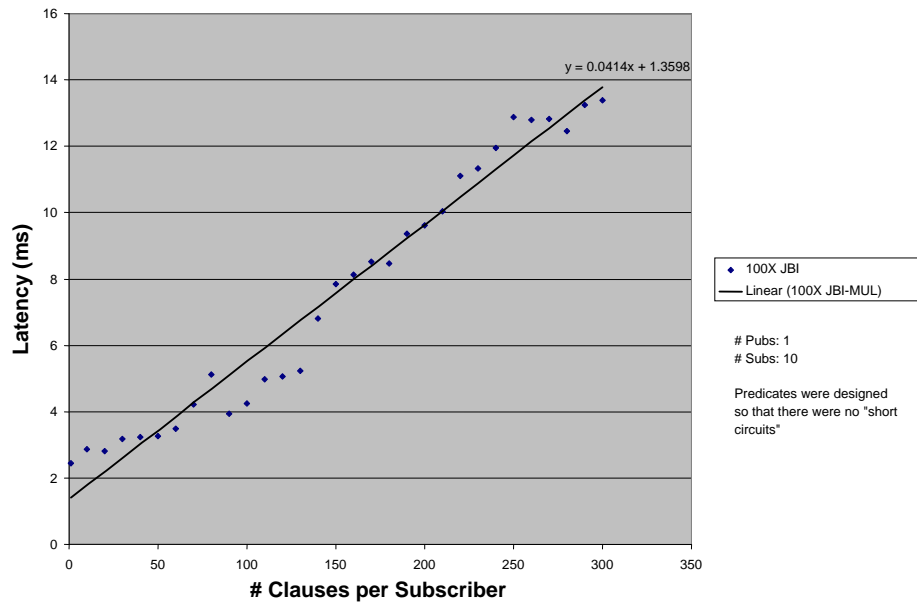


Figure 17. Predicate Complexity vs. End to End Latency

7.4 100X JBI Throughput

Previous experiments determined the publisher catcher's Information Object handling rate, which was 26 Mbytes per second on the HHPC. In addition, to ensure that the disseminator was not the bottleneck, the predicates used guaranteed that a specific information object only met one of the subscriber's requirements. Therefore, only one Information Object needed to be delivered by the disseminator.

7.4.1 Experiment

Since the publisher catcher and the disseminators were assured to handle the requirements of the test, the 1 publisher catcher, n broker and 1 disseminator configuration of the 100X JBI server was used for this experiment (Figure 18), where n is the number of brokers. With this configuration, the pub catchers and disseminators were placed on the same node and, except for the single broker configuration; all of the brokers were placed on other nodes, such that each processor in a node hosted one broker. The actual number of nodes used is shown in Table 8. The number of subscribers was set at 300, and each subscriber had a two clause predicate, which resulted in a 600 clause brokering job. The number of clients (publishers) varied depending on the desired publication rate. In Table 9 is presented the experimental parameters to determine the maximum throughput rate.

Table 8. Throughput Parameters

Computer	Coyote
Processor speed	3.06 GHz
Predicates	300
Clauses/predicate	600
Publishing Processors	4-6
Publisher Catcher Processors	1
Disseminator Processors	1
Broker Processors	1 to 16
Subscribers	300

Table 9. Throughput Experimental Conditions

Run	broker processors	pub catcher processors	disseminator processors	nodes
1	1	1	1	1
2	2	1	1	2
3	4	1	1	3
4	8	1	1	5
5	16	1	1	9

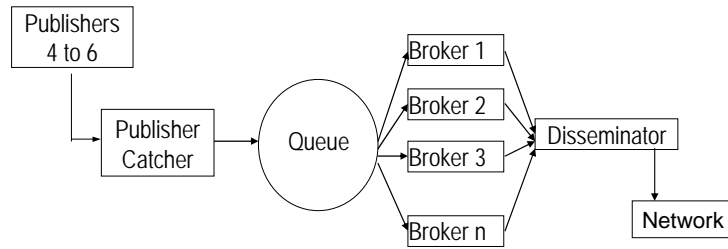


Figure 18. Architecture for Throughput of 100X JBI.

7.4.2 Results

In Figure 19 are presented the results of the 100X JBI throughput experiments. The maximum throughput rates achieved were 534, 1033, 2032, 3939, and 7036 information objects per second for 1, 2, 4, 8 and 16 brokers, respectively.

In Figure 20 is presented the maximum number of information objects that were processed in one second and the optimum number of information objects that could be processed per second verses the number of brokers used. The 100X JBI implementation was at 92% optimum for the 8 processor broker configuration, and 83% optimum with 16 brokers.

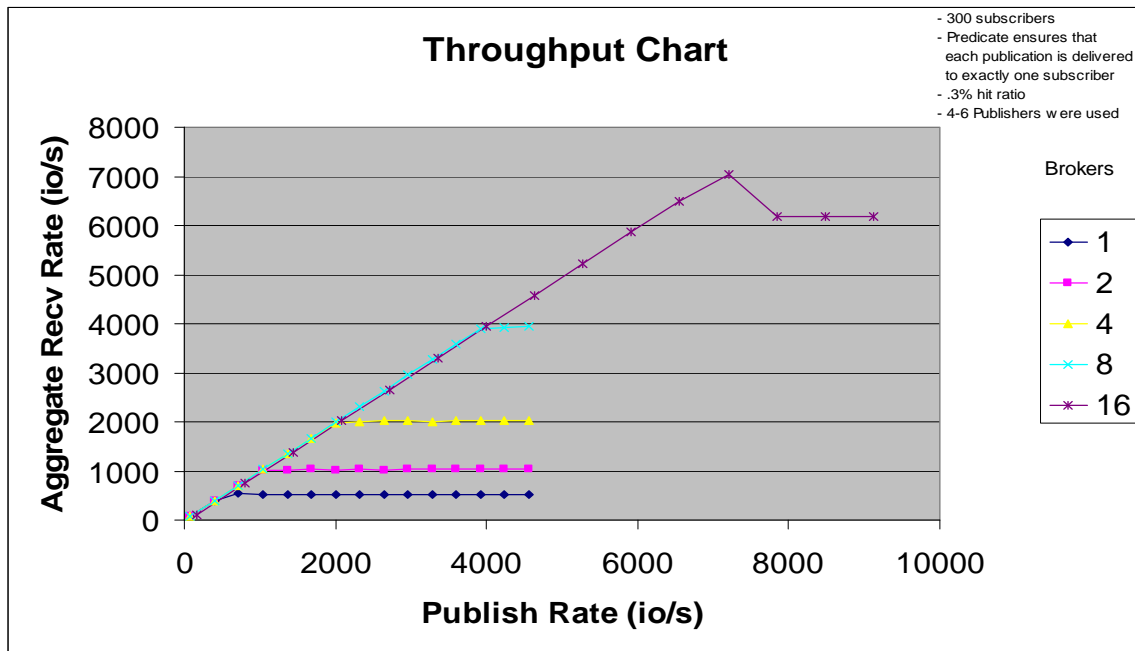


Figure 19. 100X JBI Throughput Results

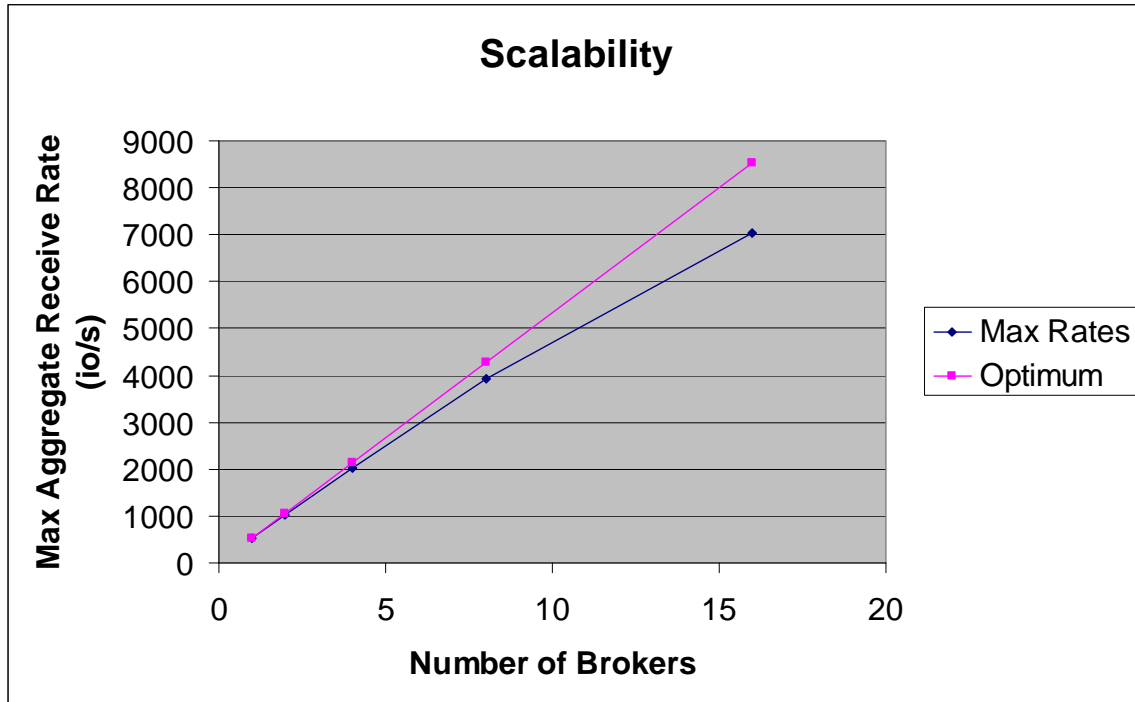


Figure 20. 100X JBI scalability

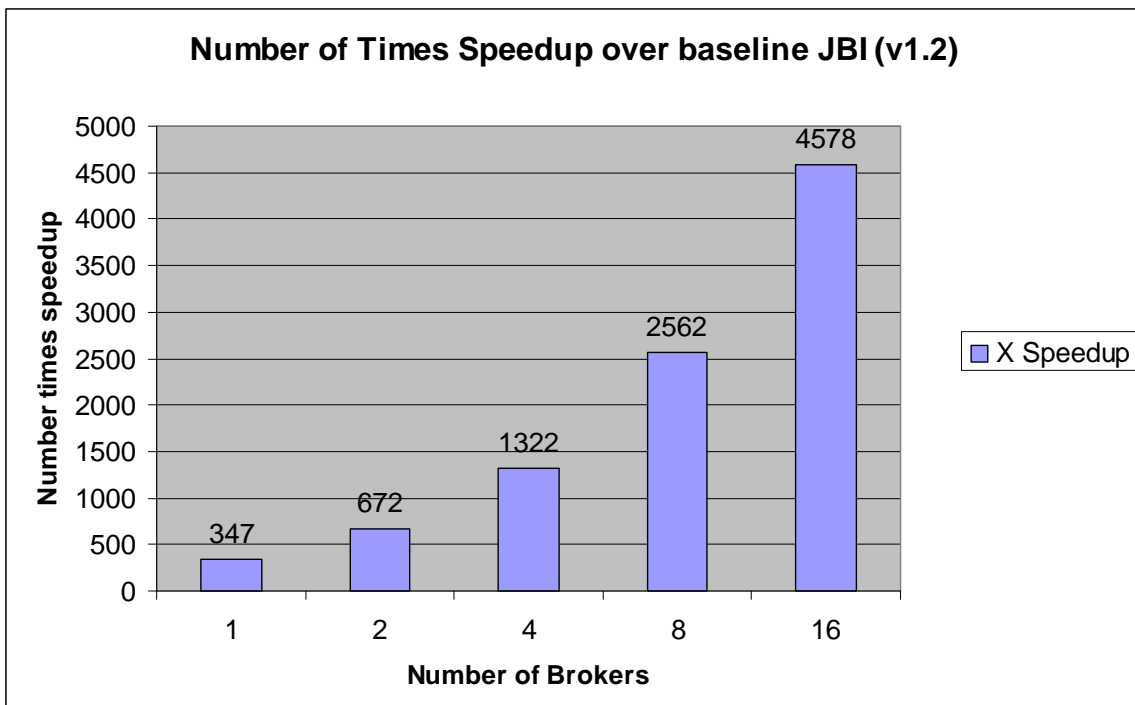


Figure 21. Speedup of the 100X JBI verses the JBI 1.2 Reference Implementation

When compared to the original JBI Reference Implementation 1.2 (Figure 21), there was a total speed up of 347 times in terms of a single broker for the 100X JBI Implementation, and 4578 times with the 16 broker configuration. The speedup of the one broker system came from a combination of rewriting parts of the code that were originally written in JAVA in C and C++, and from other code accelerations. Additional acceleration was achieved by the parallelizing the code such that several processors were simultaneously implementing the code.

8. ALPHA TESTING OF 100X JBI

The purpose of testing is to find faults, and then by correcting those faults the overall quality of the software product is improved. The usability attributes of the 100X JBI system were independently tested in terms of the ease of installation, the adequacy of the documentation, and the ease of operation. Additionally, the performed alpha test evaluation focused special attention to security-related problems to assess the degree to which the 100xJBI implementation has matured.

8.1 Overview

Testing was performed using a variety of tools and approaches to maximize both depth and coverage within an aggressive schedule and limited budget. First, commercial off the shelf (COTS) tools were utilized for static code analysis and memory profiling. These tools provided excellent coverage at minimal cost, but only allowed validation against a set of specialized known problems.

Next, a series of stress tests were performed that simulated conditions typically found in operational environments, which tended to be less controllable compared to lab environments. Tests in this category involved studying the impact of loaded computer processor unit (CPU), network links and disks on critical 100xJBI functionality. These tests were narrower in scope compared to static code checks and memory profiling.

Finally, sets of direct attacks against the system were developed ranging from attacks launched with only network layer access to attacks that assumed corrupted clients. The set of attacks explored in this thread was drawn from attack use cases developed during the 2005 red team exercises under the OASIS Dem/Val program to test the JBI Reference Implementation, and reflected conditions expected to be found in operational environments. These attacks were highly focused on exploiting single vulnerabilities and therefore provided the least amount of coverage but the most amount of depth.

8.2 Testing Parameters

The software and hardware configurations for alpha testing, and the software testing tools, are presented.

8.2.1 Software Configuration

The alpha testing was based on the following software and system configuration:

- Software Name: 100XJBI
- Software Version: 070203_3 released on May 15 2007
- Software Description: This release contained source code, executables, and documentation for the 100xJBI C++ implementation of AFRL's Joint Battlespace
- Infosphere concept. This version of the 100xJBI was compliant with the JBI Common API (CAPI) version 1.2.6

8.2.2 Hardware Configuration

For the purpose of testing the 100xJBI system, a small test bed was established that consisted of 3 Linux servers connected via a standard network switch. All 3 servers had identical hardware characteristics:

- CPU: Intel(R) Pentium(R) 4 CPU 2.66GHz
- Total Memory: 1032920 kB
- Network card: Intel Corporation 82546EB Gigabit Ethernet Controller

In addition, all 3 servers were installed with recently updated versions of the Fedora Core release 6 Linux Distribution.

8.2.3 Software Testing Tools

This section describes the set of tools used during testing together with download information for open-source tools.

- FlawFinder is a program that examined source code and reported possible security weaknesses ("flaws") sorted by risk level. It was very useful for quickly finding and removing at least some potential security problems before a program is widely released to the public. (<http://www.dwheeler.com/flawfinder/>)
- Rough Auditing Tool for Security (RATS) is an open source tool developed and maintained by Secure Software security engineers. Secure

Software was recently acquired by Fortify Software, Inc. RATS is a tool for scanning C, C++, Perl, PHP and Python source code and flagging common security related programming errors such as buffer overflows and Time Of Check, Time Of Use (TOCTOU) race conditions.

(<http://www.fortifysoftware.com/security-resources/rats.jsp>)

- Wireshark is the world's most popular network protocol analyzer. It has a rich and powerful feature set and runs on most computing platforms including Windows, OS X, and Linux. Network professionals, security experts, developers, and educators around the world use it regularly. It is freely available as open source, and is released under the GNU General Public License. (<http://www.wireshark.org/>)
- Valgrind is an award-winning suite of tools for debugging and profiling Linux programs. With the tools that come with Valgrind, many memory management and threading bugs can be automatically detect, avoiding hours of frustrating bug-hunting while making programs more stable. Detailed profiling can be preformed to speed up and reduce memory use. The Valgrind distribution currently included four tools: a memory error detector, a cache (time) profiler, a call-graph profiler, and a heap (space) profiler. It ran on the following platforms: X86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux. Valgrind is Open Source / Free Software, and is freely available under the GNU General Public License. (<http://valgrind.org/>)
- Mudflap is a pointer use checking technology based on compile-time instrumentation. It transparently adds protective code to a variety of potentially unsafe C/C++ constructs that detect actual erroneous uses at run time. The class of errors detected includes the most common and annoying types: NULL pointer dereferencing, running off the ends of buffers and strings, leaking memory. Mudflap has heuristics that allow

some degree of checking even if only a subset of a program's object modules are instrumented.

- (http://gcc.gnu.org/wiki/Mudflap_Pointer_Debugging), and
- (<http://gcc.fyxm.net/summit/2003/mudflap.pdf>)
- Mpatrol is a link library that diagnoses run-time errors caused by the wrong use of dynamically allocated memory, including writing to free
- memory and memory leaks.
- (<http://www.cbmamiga.demon.co.uk/mpatrol/>)
- Iproute2 is usually part of a package called iproute or iproute2, and consists of several tools, of which the most important are ip and tc. ip controls IPv4 and IPv6 configurations, and tc stands for traffic control. (<http://linux-net.osdl.org/index.php/Iproute2>)
- Cpuburn is designed to heavily load CPU chips. Under cooled, over clocked or otherwise weak systems may fail causing data loss (file system corruption) and possibly permanent damage to electronic components. Use this program at your own risk.
- (<http://linux.softpedia.com/get/System/Diagnostics/cpuburn-1407.shtml>)
- Bonnie++ is a benchmark suite that is aimed at performing a number of simple tests of hard drive and file system performance.
- (<http://www.coker.com.au/bonnie++/>)
- Iperf is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss. (<http://dast.nlanr.net/Projects/Iperf/>)
- Netcat is a featured networking utility which reads and writes data across network connections, using the TCP/IP protocol. It was designed to be a reliable "back-end" tool that can be used directly or easily driven, by other

programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection needed and has several interesting built-in capabilities. (<http://en.wikipedia.org/wiki/Netcat>)

- GDB, the GNU Project debugger, allows the user to see either what is going on 'inside' another program while it executes, or what another program is doing at the moment it crashed. (<http://sourceware.org/gdb/>)

8.3 Summary of the Testing Results

The first direct result from testing effort was that the software was successfully installed and ran the supplied test applications, which performed publish, subscribe, and query operations. Deploying the 100xJBI outside of its lab environment was a major step towards TRL 6 compliance, and it was discovered that there were a number of issues of varying severity during the course of testing the system. The documented details of these issues resulted in 57 problem tickets, which fell into the following main categories:

8.3.1 Installation Testing

The installation procedure was quite complex. The installation scripts supplied with the 100X JBI release provided insufficient documentation on main concepts (e.g., difference between cluster and single node install, Network File System (NFS) shares), 3rd party code requirements, and the relationship of the 100X JBI to the JBI Reference Implementation.

Although the 100X JBI provided scripts for installing and starting the system, it didn't provide adequate monitoring capabilities to ascertain successful operations of the overall system. There was no monitoring protocol to test the liveliness of processes. An accidental crash of a PubCatcher process could lead to situations in which a PubCatcher may remain in a crashed state for an extended period of time during low-usage phases, which only would be noticed (reactively) when critical operations started failing.

A proactive monitoring protocol could be developed to detect process crashes shortly after happening, and the processes could be quickly restarted (either automatically or per human intervention) to recover from the outage before the system needed to

service the next critical operation. There was no tool support for clearing out payload entries stored in the filestore directory on the NFS share.

In response to this, a documentation package was created that included install directions, a 100X JBI client tutorial, a 100X JBI client API and platform specific documentation.

8.3.2 Static Code Analysis and Memory Profiling

Numerous potential security vulnerabilities were tested for that could be exploited to crash components and circumvent security measures. These issues have either been addressed and fixed, have yet to be addressed but were noted, or were outside the scope of this effort, but also have been noted. .

Memory leaks and corruption errors were identified in both the client-side CAPI implementation and core-resident PubCatcher processes. Most worrisome in this category were the memory corruption errors in the PubCatcher process, as it could be exploited to develop buffer overflow attacks to take control of machines running the PubCatcher processes. The major leaks were addressed as each was reported.

There were single points of failure for various components. Although all runtime testing was performed in a cluster configuration with three servers, crashes of a single process or corruption of single files directly caused a loss of availability and confidentiality for various different components. MySQL single point of failure vulnerability in the cluster configuration was identified. The crash of a single PubCatcher process resulted in a direct failure of critical functionality to publish information objects. BerkleyDB single point of failure vulnerability was also identified, as was the NFS server that exported the share for storing the BerkleyDB data. This is a result of the tools that the server was built with. Currently, there are no plans to migrate to a different set of tools.

There were inconsistencies with the CAPI semantics. Improper exception semantics were identified during testing which was likely introduced through by caches within the 100X JBI. Furthermore, IOs with incorrect size values were silently being dropped. This raised issues with the C++ interface of the CAPI in terms of requiring an

explicit specification of the payload size as well as the absence of exceptions when IOs were being dropped. This problem was noted, and will be considered at a later time.

8.3.3 Stress Testing

Subjecting the system to an environment with increased CPU, disk, network, and application usage loaded that the PubCatcher file, which caused the PubCatcher to lock after 1024 client connects. Also, creating multiple (successive) publisher sequences and their activation caused a large latency variance. There is a limitation of how many clients can be connected to a pubcatcher. The system has to be configured according to the number of pubcatchers to handle the desired number of clients.

The crash of a single PubCatcher process resulted in direct failure of critical functionality to publish information objects. Part of this problem stemmed from the inherent failure semantics that come with Message Passing Interface – 2 (MPI-2). While it may make sense to terminate all processing upon observing a single process crash in a scientific computation environment, such semantics are not desirable for a 100X JBI that services real-time mission critical applications that needs to continue to provide service even under attack. This is a design flaw in MPI, in which if one part of an application ceases to function, the whole application closes.

Currently, MPI is the community standard, and there are no immediate plans to migrate to a different library for the 100X JBI, although alternatives to MPI are being explored in other on-going projects. Interestingly enough, the 100X JBI is one of the information management systems being evaluated as a possible alternative for MPI for uses such as this. The number of publishers is independent of the number of subscribers in this paradigm, so that failure or addition of nodes does not affect its operation.

8.3.4 Direct Attacks

Running COTS attacks against the PubCatcher affected critical functionality. Also, TCP connection floods resulted in the unavailability of the PubCatcher. Sending a random stream of bytes over a single connection to port 11011 of the PubCatcher caused legitimate publish functionality to be blocked. Rogue clients could also flood the system with a large number of IO's and deny service to legitimate clients. These issues have not yet been addressed.

Weaknesses in handling of sensitive password information and policy settings were also observed. MySQL passwords were stored in clear text in ConnectionService.cnf, which is world readable and stored on an NFS share. Also, privilege separation established by the JBI reference implementation was weak in that the cmp user could change MySQL access tables. These issues were raised in the documentation and rely on the administrators' knowledge of MySQL and UNIX to secure the installation.

8.4 Towards Beta Testing

The purpose of testing is to find faults, which can then be corrected. The results and recommendations of the Alpha Testing are being evaluated, and necessary changes are being incorporated into the 100X JBI software. Some of the changes are being recommended to be incorporated into the JBI Reference Implementation, as they are generic to all of the different versions of the JBI.

9 DEMONSTRATIONS

Among the many notable demonstrations of the 100X JBI Information Management software that have been proposed include the Swathbuckler Experiment, the Angel Fire Experiment, and the Paradigms for Parallel Computing.

9.1 Swathbuckler

A prototype 100X JBI was demonstrated by the Air Force Research Laboratory as the information management system for exploiting real-time formed Synthetic Aperture Radar (SAR) images as they were acquired in an airborne experiment. The prototype was used for two information management systems, one for the embedded system and one for the data repository.

All communications between clients were defined by XML schemas, which allowed clients to be developed separately and led to rapid prototyping and deployment. The Swathbuckler user console provided simultaneous access to performance chart displays, algorithmic statistics, meta-data moving maps, and image display interfaces. Additionally control of the mission data, data collection types, and history were provided. All the nodes regularly published status and subscribed to commands utilizing the prototype.

The security provided by the prototype 100X JBI implementation was crucial for connecting the remote user to the real-time system. The prototype used private networking including tunneling technology to allow communication connections over insecure networks. In this case a JBI client connected from the airplane over the internet to a JBI server at a remote location, and other users to connect in real time to communicate and receive information from the airplane.

9.2 Angel Fire

The 100X JBI has been proposed as an information management system for the Angel Fire, a Los Alamos National Laboratory/Air Force Research Laboratory persistent city-sized surveillance program. Angel Fire is in fact an airborne high-resolution imaging and dissemination system, and provides real-time imaging capabilities. The total

infrastructure would enable users to quickly and easily sort through hundreds of terabytes of image data to publish important metadata imagery acquired at a specific time, the ability to subscribe to imagery and metadata regarding specific locations, and the ability to overlay video over maps.

9.3 Paradigms for Parallel Computing

As briefly discussed in section 8.3.3 of this report, the 100X JBI software, and the developing 100K JBI software, is being evaluated at Arizona State University by Dan Stanzione for the User Productivity Enhancement and Technology Transfer (PET) Program of the DoD's High Performance Computing Modernization Program. In particular, the service – orientated approach provided by the 100X JBI, and its fault tolerant attributes inherent in a publish-subscribe information management, make this in principle an ideal system for passing information between hundreds to thousands of nodes.

10 YFILTER BROKERING

The YFilter is a system designed for XML brokering task. In this system when a predicate is processed, the resultant products are shared with as many other processors as possible, thus reducing the amount of redundant processing between predicates as possible. This sharing reduces the total time required to evaluate all predicates for a given Information Object (IO). The concepts from the YFilter were implemented and extended in the C++ language, thus increasing the processing speed from the YFilter's native implementation. The resulting reduction in brokering time can enable greater scalability in the JBI.

The YFilter Broker has been integrated with the 100X JBI system, and in that implementation the YFilter was the default broker, and was tested on the AFRL Coyote High Performance Computing cluster. The improvements described in this paper yielded up to a 15 fold decrease in brokering latency and up to a 15 fold increase in system throughput when compared with the prior software broker. A report titled "Using YFilter Concepts for Fast Brokering in the JBI" authored by Justin M. Fiore, Lei Zhao and Vincent J. Mooney III of the Georgia Institute of Technology is being published as a separate government report.

11. SUMMARY

The results of the experiments designed to evaluate the core service speedups of the 100X JBI architecture have shown speedups of up to 4578 times verses the JBI reference implementation. The resultant 100X JBI system has been alpha tested, faults uncovered by that testing are being corrected, and beta testing will occur under the Real Time Infospaces effort.

The 100K Infosphere effort, another ongoing effort, will integrate together the relevant parts of the reference implementation, the 100X JBI system, the field programmable gate array efforts, and the YFILTER results, to speed up the core services of the 100X JBI by 5 orders of magnitude.

12 REFERENCES

1. AFRL/IFSE, Reference Implementation Quick Start Guide, Core Services Reference Implementation Version 1.2.6, 14 Dec 2005.
2. Hatch, Brian, "SSH Port Forwarding." Infocus, 06 January 2005 (<http://www.securityfocus.com/infocus/1816>).

APPENDIX Symbols, Abbreviations and Acronyms

100X	One hundred times speedup
AFRL	Air Force Research Laboratory
C++	A general purpose computer programming language. Originally known as <i>C with Classes</i>
CAPI	Common Application Programming Interface
http	HyperText Transfer Protocol
IOR	Information Object Repository
IMS	Information Management Staff
IP	Internet Protocol
JB1	Joint Battlespace Infosphere
JMS	Java Messaging Services
JNI	Java™ Native Interface is a standard programming interface for writing Java native methods and embedding the Java™ virtual machine into native applications. The primary goal is binary compatibility of native method libraries across all Java virtual machine implementations on a given platform.
JniConnection	JniConnection is the Java Native Interface (JNI) wrapper to call our c++ JB1 client libraries from java
JniConnectionService	The Java Native Interface C++ code that enables 100X JB1 clients in C++ or Java to communicate to the original JB1 Java server code that provides the security framework.
MDR	Metadata Repository
MPI	Message Passing Interface
POST	Submits user data (e.g. from a HTML form) to the identified resource. The data is included in the body of the request.
RBAC	Role Based Access Control
SSH	Secure Shell (protocol)
SSL	Secure Socket Layer
VPN	Virtual Private Network

XML	Extensible Markup Language
XMP	Extensible Metadata Platform
XPATH	XMP Path Language
YFilter	A single Nondeterministic Finite Automaton which combines multiple queries into a single query.